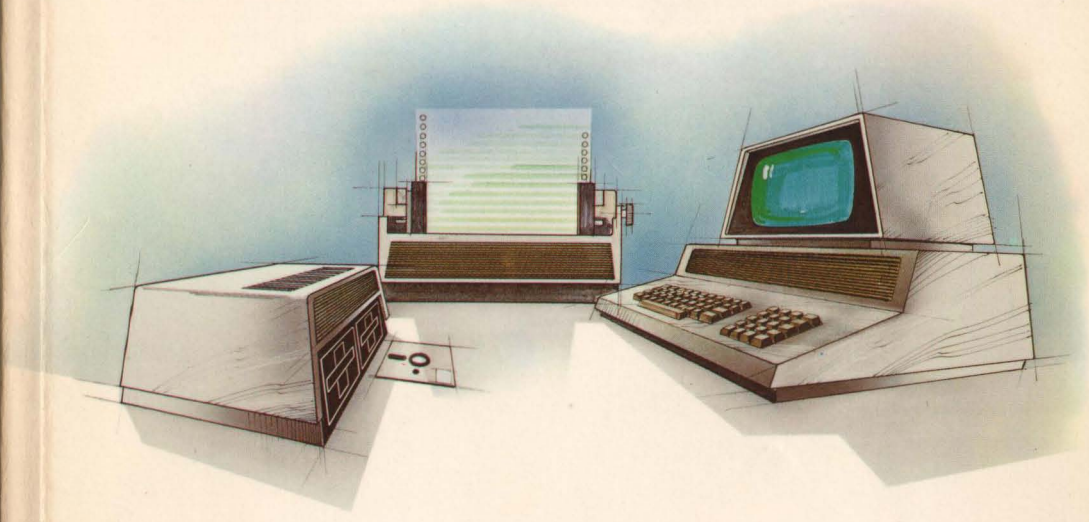
CBM Computer
CBM Bandgeräte

CBM Floppy-Disks
CBM Textdrucker

CBM COMPUTER HANDBUCH



Adam Osborne

Carroll S. Donahue

te-wi

CBM Computer
CBM Bandgeräte

CBM Floppy-Disks
CBM Textdrucker

CBM COMPUTER HANDBUCH

Adam Osborne

Carroll S. Donahue

te-wi

IMPRESSUM

Grundlage dieses Buches ist die Übersetzung der amerikanischen Originalausgabe "PET/CBM Personal Computer Guide, Second Edition" by Adam Osborne and Caroll S. Donahue.

Die Übersetzung wurde mit dankenswerter Unterstützung von COMMODORE BÜROMASCHINEN GMBH für den deutschen Sprachraum völlig überarbeitet und auf den neuesten Stand gebracht.

©Copyright 1980 by McGraw-Hill, Inc.

Alle Rechte vorbehalten. Ohne ausdrückliche, schriftliche Genehmigung der Herausgeber ist es nicht gestattet, das Buch oder Teile daraus in irgendeiner Form durch Fotokopie, Mikrofilm oder ein anderes Verfahren zu vervielfältigen oder zu verbreiten.
Dasselbe gilt für das Recht der öffentlichen Wiedergabe.

Übersetzung ©Copyright 1981 by McGraw-Hill, Inc.

LIZENZNEHMER und HERAUSGEBER:
te-wi Verlag GmbH, Theo-Prosel-Weg 1
8000 München 40, Telefon 089/192090

Die Herausgeber übernehmen keine Gewähr dafür, daß die beschriebenen Schaltungen Baugruppen, Verfahren usw. funktionsfähig und frei von Schutzrechten Dritter sind.

CBM ist Warenzeichen von Commodore Business Machines

GESAMTHERSTELLUNG:
technik marketing, München
tm 3403/283 3.Auflage
Printed in Germany
ISBN 3- 921803-13-6

Vorwort

Dieses Buch beschreibt alle CBM-Computer: Den ursprünglichen CBM 2001 Personal Electronic Transactor, mit dem CBM berühmt wurde, und die danach eingeführten Serien CBM 3001, CBM 4001 und, als letztes Modell, den CBM 8032 mit einem 80spaltigen Bildschirm. Weiterhin wird ausführlich das Arbeiten mit folgenden CBM-Peripherie-Geräten beschrieben: Bandkassette CBM C2N, Floppy Disk-Speicher CBM 3040, CBM 4040 und CBM 8050 und Textdrucker CBM 3022, CBM 4022 und CBM 8024.

Ebenfalls werden die neuesten Software-Entwicklungen von CBM beschrieben: BASIC 4.0, die neueste Version der CBM BASIC-Programmiersprache, und zwei neue Versionen des Betriebssystems für Floppy Disk-Speicher, DOS 2.1 und DOS 2.5, in der Literatur unter DOS 2.0 bekannt.

Die Darstellung der Programmierung in der Programmiersprache BASIC wurde wesentlich erweitert. Selbst wenn Sie nie an einem Computer gesessen und programmiert haben, wird Sie dieses Buch befähigen, eigene BASIC-Programme für Ihren CBM-Computer zu schreiben.

Dieses umfangreiche Buch enthält eine fast erschöpfende Darstellung der CBM-Computer und ihrer Peripherie. Möglicherweise benötigen Sie für Ihre Aufgabe nur einen Teil dieser Informationen.

Vielleicht wollen Sie gar kein Programmierer werden. In Kapiteln 1 und 2 erfahren Sie alles über den Umgang mit CBM-Geräten, um bereits geschriebene Programme auf Ihrer Anlage ablaufen zu lassen. Sie können den Rest des Buches übergehen, bis es Sie eines Tages reizt, eigene Versuche in der Programmierung Ihres CBM-Computers zu beginnen.

Kapitel 4, 5 und 8 zeigen den Gebrauch der Programmiersprache CBM BASIC. In Kapitel 8 finden Sie eine vollständige, komprimierte Darstellung aller in CBM BASIC verfügbaren Befehle. Kapitel 4 gibt eine Einführung in die Programmiersprache BASIC, während Kapitel 5 an praktischen Beispielen zeigt, wie umfangreiche Aufgaben in CBM BASIC formuliert und gelöst werden.

In Kapitel 6 wird Ihnen vorgeführt, wie Sie in der Programmiersprache CBM BASIC den Datenverkehr zwischen Ihrem Computer und einem Bandgerät, einem Floppy Disk-Speicher oder einem Drucker abwickeln.

Kapitel 7 enthält System-Informationen für den Experten. Die Informationen in diesem Kapitel werden für Sie wertvoll, wenn Sie über die Möglichkeiten von CBM BASIC hinausgehen und z.B. in ASSEMBLER-Sprache programmieren wollen.

Keine Unterweisung im Programmieren ist so eindrucksvoll wie ihre Darstellung an Beispielen. Dieses Buch ist voller kurzer Programmbeispiele. Geben Sie die Programmbeispiele beim Lesen des Buches in Ihren Computer ein, und lassen Sie sie ablaufen. Jedes Programmbeispiel dieses Buches ist zuvor auf einem CBM-Computer abgelaufen. Aber nur ein überheblicher Programmierer kann behaupten, daß alle seine

Programme völlig fehlerfrei sind. Wenn eines der Programme in diesem Buch nicht auf Ihrem Computer ablaufen will, dann denken Sie an folgende mögliche Fehlerquellen: Den Urheber des Programms, veränderte Bedingungen des Programmablaufs auf Ihrem Computer und: Eingabefehler. Bitte teilen Sie uns unerwartete Erlebnisse mit den Programmbeispielen dieses Buches mit.

Auf allen CBM-Computern stehen Ihnen zwei Zeichensätze zur Verfügung: Der Standard-Zeichensatz und der Alternativ-Zeichensatz. Wir haben durchgehend in diesem Buch alle Programme mit den Zeichen des Standard-Zeichensatzes geschrieben. Wenn Sie die Programmbeispiele in der gleichen Form auf Ihrem Computer wiederholen wollen, stellen Sie sicher, daß Sie Ihren Computer auf den Standard-Zeichensatz umgeschaltet haben. Derartige Umschaltungen finden Sie ausführlich in Kapitel 1 beschrieben.

Dieses Buch wird den verschiedenen Versionen der Programmiersprache CBM BASIC gerecht: BASIC 1.0, BASIC 3.0 und BASIC 4.0. Die bedeutendsten Erweiterungen weist die Version CBM BASIC 4.0 auf. Daher teilen wir die Beschreibungen dieses Buches in die Gruppen BASIC 4.0 und BASIC < 3.0 (für alle früheren Versionen von BASIC 4.0).

Die verschiedenen Versionen des Betriebssystems für Floppy Disk-Speicher, DOS, beeinflussen zwar die Programmierung, jedoch weit weniger als die Verschiedenheiten zwischen den Versionen von CBM BASIC. Wir erwähnen daher durchgehend in diesem Buch auftretende Verschiedenheiten zwischen den Versionen von CBM BASIC, jedoch seltener Verschiedenheiten zwischen den Betriebssystemen.

Inhalt

1. Einführung in die CBM-Computer-Systeme 1-1

Komponenten eines Computer-Systems 1-2

CBM-Computer 1-4
Eigenschaften von CBM-Computern. Bildschirme. Tastaturen. Zeichensätze. Editionsmöglichkeiten.

CBM-Speicher 1-27
CBM-Bandgerät C2N. Externer Anschluß. Bandkassetten. Arbeiten mit dem Bandgerät. **CBM Floppy Disk-Speicher.** Technische Daten. Anschluß. Umgang mit den Disketten.

CBM-Textdrucker 1-40
Technische Daten. Schriftbilder. Anschluß. Einlegen von Papier und Farbband. Funktionstest.

2. Sofort arbeiten mit CBM-Geräten 2-1

Direkter Dialog 2-1
Zulässige Eingaben. Anweisungen zur Bildschirmausgabe. Arithmetische Berechnungen. Bewegung des Kursors.

Programmform 2-10
Programmauflistung. Programmstart. Programmbeendigung. Programmlöschung.

Verfügbare Zeichensätze 2-14

Arbeiten mit dem Bandgerät 2-15
Abspeichern eines Programms. Laden eines Programms. Laden und Ablauf eines Programms.

Arbeiten mit dem Floppy Disk-Speicher 2-19
Betriebssystem DOS für Floppy Disk-Speicher. Laden eines Programms von Diskette in BASIC < 3.0. Öffnen einer logischen Datei. Initiieren eines Laufwerks. Inhaltsverzeichnis einer Diskette. Anlegen einer neuen Diskette. Abspeichern eines Programms. Laden eines Programms von Diskette in BASIC 4.0. Inhaltsverzeichnis einer Diskette. Anlegen einer leeren Diskette. Abspeichern eines Programms.

Arbeiten mit dem Textdrucker 2-29
Öffnen eines Datenkanals. Datenübertragung zum Drucker. Schließen des Datenkanals.

3. Textbearbeitung am Bildschirm 3-1

Direkte Textkorrektur am Bildschirm 3-1
Textlöschung mit der Taste DEL. Texteinfügung mit der Taste INST.

Textkorrektur zwischen "Anführungszeichen" 3-6

Edition von Programmanweisungen 3-9
Duplizieren von Programmzeiten. Edition ähnlicher Programmanweisungen

Bildschirmedition in BASIC 4.0

4. Programmierung von CBM-Computern 4-1

Direkter Dialog und Programmform 4-1

1zeilige Programme in Direktem Dialog 4-3

ABC der Programmiersprache CBM BASIC 4-6
Zeilennummern. Darstellen von Daten. Zeichenketten. Variable. Variablennamen

Operatoren 4-15
Arithmetische Operatoren. Vergleichs-Operatoren. Logische Operatoren.

Tabellen (Felder) 4-23

BASIC-Befehle 4-25

Programmverzweigungen 4-31
Programmschleifen. Unterprogramme.

Ein-/Ausgabe-Anweisungen 4-40
Datenausgabe mit PRINT.

Bildschirmgestaltung 4-42
Spaltenfunktion SPC. Tabulierfunktion TAB.

Funktionen 4-47
Arithmetische Funktionen. Zeichenketten-Funktionen. System-Funktionen. Benutzer-Funktionen.

5. Wahrnehmen aller Fähigkeiten eines CBM-Computers 5-1

Gerätetechnische Fähigkeiten 5-1
Tastenfeldspeicher.

Verbinden von Zeichenketten 5-5
Verkettung bei Drucker/Bildschirmausgaben. Graphische Zeichenketten. Numerische Zeichenketten.

Ein-/Ausgabe-Programmierung 5-9
Möglichkeiten des PRINT-Befehls.

Bewegung des Kursors 5-15

Zeicheneingabe im ASCII-Kode 5-18

Eingabemöglichkeiten an CBM 8032-Computern 5-19

Zeichenausgabe auf bestimmte Rasterpunkte des Bildschirms 5-22

Aufbau guter Programme zum Einlesen von Daten 5-24
Interaktive Dateneingabe. Eingabe ähnlich einer Formblatteintragung.

Programmierung von Bildschirm- und Druckerausgaben 5-41

Rechnen mit mehr als 9stelligen Zahlen 5-47
Addition. Subtraktion. Multiplikation.

Computer-Graphiken 5-76
Reproduktion von Graphiken. Belebte Graphiken.

Die Echtzeit-Uhr im CBM-Computer 5-85

Zufallszahlen 5-91
Startzahlen von Zufallsfolgen. Bildschirmdarstellung von Zufallszahlen. Ausdrucken von Zufallszahlen. Ziehen zufälliger Startzahlen. Zufallszahlen in beliebigem Wertebereich. Zufallsgesteuerte Symboldarstellungen auf dem Bildschirm.

6. Peripherie-Geräte zu CBM-Computern: Datasette-Bandgerät, Floppy Disk-Speicher und Drucker 6-1

Datenspeicherung auf magnetischen Trägern 6-2

Informationsaustausch zwischen Kassette/Diskette und Computer 6-4

Arbeiten mit Dateien auf Kassetten 6-11
Öffnen einer Datei. Schließen einer Datei. Zugriff auf Bandkassetten-Dateien. Speichern von Zahlen. Speichern von Zeichenketten. Lesen von Daten. Organisationsformen von Banddateien.

Arbeiten mit Dateien auf Disketten 6-41
Datenspeicherung auf Disketten. Relative Dateien. Sequentielle Dateien. Bildung von Dateinamen. Versionen des Betriebssystems DOS. Versionen von CBM BASIC.

Öffnen einer Disketten-Datei 6-50

Schließen einer Disketten-Datei 6-52

Fehlermeldung und Gerätestatus bei Floppy Disk-Speichern 6-53

Verwaltung von Disketten-Dateien 6-55
Anlegen und Initiieren einer Diskette. Ausgabe des Dateien-Verzeichnisses. Inventur eines Dateien-Verzeichnisses. Überspielen von Dateien und ganzen Disketten. Neubenennung einer Datei. Löschen von Dateien.

Sequentielle Dateien für Daten 6-65
Trennsymbole für Felder. Schreiben numerischer Daten. Schreiben von Zeichenketten. Anfügen von Daten.

Relative Dateien in BASIC 4.0 6-78
Trennsymbole zwischen Feldern. Lesen von Aufzeichnungen. Schreiben numerischer Daten. Schreiben von Zeichenketten. Direkter Zugriff auf Aufzeichnungen.

Gebrauch von GET# für Disketten-Dateien 6-84

Programm-Dateien 6-87
Laden und Speichern. Programm-Warteschlangen.

Programmierung von CBM-Druckern 6-90
Daten ausdrucken wie empfangen. Formatierte Ausdrucke. Steuerzeichen. Formatierung von Druckseiten. Selbstentworfene Druckzeichen. Fehlermeldungen.

7. Informationen zum Systemaufbau 7-1

Aufbau von CBM-Computern 7-1
Speicherauszüge 7-6
CBM BASIC-Übersetzer 7-7
Speicherung von BASIC-Anweisungen 7-9
Datenformate 7-11
Speichern von Tabellen (Feldern) 7-16
Zeichenkodierungen in CBM-Computern 7-19
Programmierung in Assembler 7-20
Dateien für wahlfreien Zugriff 7-23

8. CBM BASIC 8-1

Befehle in CBM BASIC 8-2
Funktionen in CBM BASIC 8-34
CBM 8032-Editionsfunktionen 8-45

ANHÄNGE

A CBM-Zeichen-Kodes
B CBM-Fehlermeldungen
C Umwandlungs-Tabellen
D Abweichung bei Computern mit Version-2-ROMs
E Literatur

Programme

Nachweis der wichtigsten Programmbeispiele dieses Buches

		Seite
DATUM	Eingabe des Tagesdatums im Dialog Benutzer-Computer	5-34
ADRESSKARTEI	Adresseingaben im Dialog Benutzer-Computer über Formblattdarstellungen auf dem Bildschirm	5-38
LESEFENSTER	Ausschnittsweise Darstellung großer Tabellen auf dem Bildschirm mit einem verschieblichen Lesefenster	5-44
ADDITION	Additionsprogramm für bis zu 16stellige Zahlen	5-52
SUBTRAKTION	Subtraktionsprogramm für bis zu 16stellige Zahlen	5-61
MULTIPLIKATION	Multiplikationsprogramm für bis zu 16stellige Zahlen	5-74
ANIMATION	Belebte Computer-Graphik: langsam fortschreitende Darstellung einer Graphik auf dem Bildschirm	5-83
UHRZEIT (1)	Bildschirmdarstellung der Uhrzeit durch Ablesen der Computer-internen Uhr	5-86
LAUFZEIT	Messen einer Programmlaufzeit mittels der Computer-internen Stopuhr	5-88
UHRZEIT (2)	Bildschirmdarstellung der Uhrzeit in großen Ziffern	5-89
ZUFALLSZAHL (1)	Darstellung von Folgen von Zufallszahlen auf dem Bildschirm	5-92
ZUFALLSZAHL (2)	Ausdruck von Folgen von Zufallszahlen	5-94
M, N	Erzeugung von Zufallszahlen im beliebig wählbaren Wertebereich m bis n	5-95
ZUFALLSZAHL (3)	Zufallsgestreute Bildschirmausgabe beliebig wählbarer Symbole	5-98
NUM. PRINT #	Schreiben von Zahlen auf Kassette	6-18
NUM. INPUT #	Lesen von Zahlen von einer Kassette	6-25
WORD. PRINT #	Schreiben von Worten auf Kassette	6-19
WORD. INPUT #	Lesen von Worten von einer Kassette	6-25
NAME. PRINT #	Schreiben von Namen auf Kassette	6-21
NAME. INPUT #	Lesen von Namen von einer Kassette	6-21

MAIL. PRINT #	Anlegen einer Adressenkartei auf Kassette; Adresseingabe über die Tastatur des Computers	6-21
MAIL. INPUT #	Lesen einer Adressenkartei von einer Kassette; Adressdarstellung auf dem Bildschirm	6-27
MAIL. GET #2	w.o., jedoch mit der Anweisung GET# realisiert	6-31
SEQ. NUM.	Schreiben numerischer Daten in eine Sequentielle Disketten-Datei	6-66
SEQ. ADR. LISTE.	Schreiben einer Adressenkartei in eine Sequentielle Disketten-Datei	6-70
SEQ. NUM. APPEND	Anfügen von Daten an eine bestehende Datei mit APPEND#	6-74
SEQ. TEST. CONCAT	Verkettung Sequentieller Disketten-Dateien	6-76
REL. NUM.	Schreiben numerischer Daten in eine Relative Disketten-Datei	6-80
REL. ADR. LISTE.	Schreiben einer Adressenkartei in eine Relative Disketten-Datei	6-81
NUM. FORM. PRINT.	Formatierter Ausdruck von Zahlen	6-96
PRINT. FORM. ZEICHK.	Formatierter Ausdruck von Zeichenketten	6-102
PRINT. DATUM	Formatierter Ausdruck des Tagesdatums	6-104
PRINT. FORM. ZEICHK. 2	Formatierter Adressausdruck in Breitschrift	6-105
ZEILENZAHL	Änderung der Zeilenzahl je Seite	6-108
PFUND-SYMBOL	Ausdruck des englischen Pfundzeichens als Beispiel für Benutzer-entworfene Druckzeichen	6-110

KAPITEL 1

Einführung in die CBM Computer Systeme

Im Jahre 1977 brachte Commodore Business Machines, Santa Clara, Kalifornien als erstes Tischcomputer-System die Serie CBM 2001 Personal Electronic Transactor auf den Markt. Computer dieser Serie vereinigen in kompakter Bauweise die wichtigsten Komponenten eines Computer-Systems: Den eigentlichen Computer, ein Tastenfeld für die Eingabe von Buchstaben, Zahlen, graphischen Symbolen und Sonderzeichen, einen Bildschirm zur Darstellung von Eingaben oder Antworten des Computers sowie ein Speichergerät für Programme und Daten in Form eines eingebauten Kassetten-Bandgeräts. Die Fähigkeiten der CBM 2001 Computer wurden in den Serien CBM 3001, CBM 4001 und CBM 8001 weiter entwickelt, wobei besonders folgende Entwicklungen bemerkenswert sind:

1. Die in CBM-Computern verwendete Programmiersprache CBM BASIC wurde weiterentwickelt, wodurch sich insbesondere die Dialogmöglichkeiten mit externen, an den CBM-Computer angeschlossenen Geräten erheblich erweiterten.
2. Die interne Speicherkapazität von CBM-Computern wurde vergrößert, so daß heute Modelle mit einer Speicherkapazität von 32K Byte angeboten werden. (1K Byte bedeuten die Speicherkapazität für 1024 Zeichen, aus denen ein Programm, ein Brief, eine Tabelle usw. bestehen kann).
3. Die kompakte Tastatur der CBM 2001-Serie wurde in den neuen Computer-Serien auf die Größe von Schreibmaschinen-Tastaturen gebracht und damit an kaufmännische bzw. technisch-wissenschaftliche Anwendungen angepaßt.
4. Die Bildschirmgröße bei Computern der Serie CBM 8001 wurde auf eine Zeilenbreite von 80 Zeichen verbreitert und damit insbesondere an Bedürfnisse der kommerziellen Textverarbeitung angepaßt.
5. Die Leistungsfähigkeit extern anschließbarer CBM-Speichergeräte und CBM-Textdrucker wurde erheblich vergrößert.

Neben den Computer-Serien hat Commodore Business Machines auch Serien von CBM-Speichergeräten zur externen Speicherung von Programmen und Daten sowie CBM-Textdrucker zum kommerziellen oder technisch-wissenschaftlichen Ausdruck von Daten und Programmen entwickelt.

Diese Serien von CBM-Computern, CBM-Speichergeräten und CBM-Druckern liefern alle Komponenten für die Zusammenstellung kompakter Computer-Systeme.

KOMPONENTEN EINES COMPUTER-SYSTEMS

CBM-Computer eignen sich durch ihre Leistungsfähigkeit und Anschlußmöglichkeiten zum Aufbau von Computer-Systemen, in denen eine fast unbeschränkte Vielfalt elektronischer Geräte in Dialog mit dem Computer tritt.

An erster Stelle sind Geräte zu nennen, die Leistungsfähigkeit und Funktion von CBM-Computern bedeutend erweitern: Geräte zur Speicherung großer Datenmengen (CBM Floppy Disk-Speicher, CBM-Bandgeräte) und Geräte zum schnellen Ausdrucken umfangreicher Datenmengen (CBM-Textdrucker). Figur 1-1 zeigt den typischen Aufbau eines Computer-Systems aus diesen Komponenten.

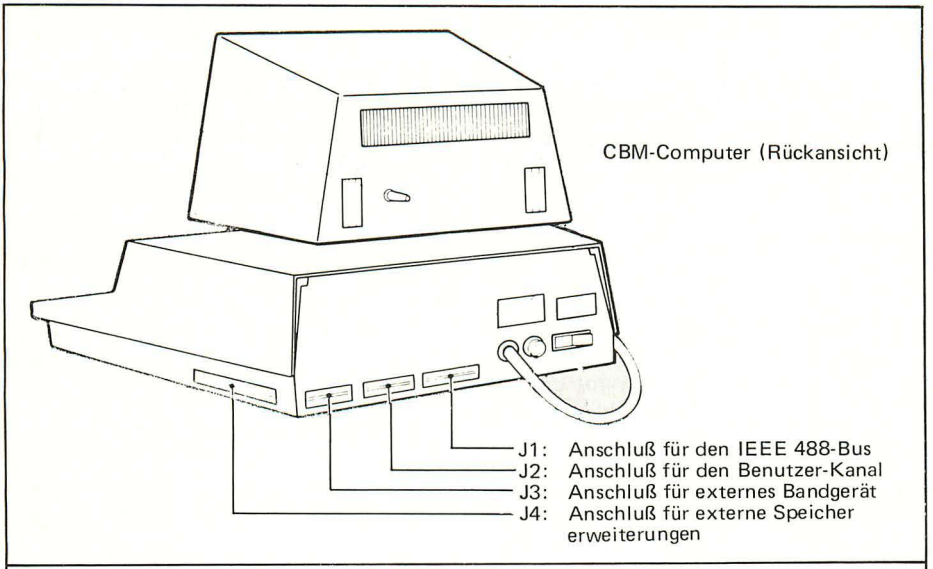
Daneben können elektronische Geräte, wie sie in Prüfeinrichtungen, Analyseaufbauten, Prozeßsteuerungen usw. auftreten, für einen programmgesteuerten Dialog an CBM-Computer angeschlossen werden.

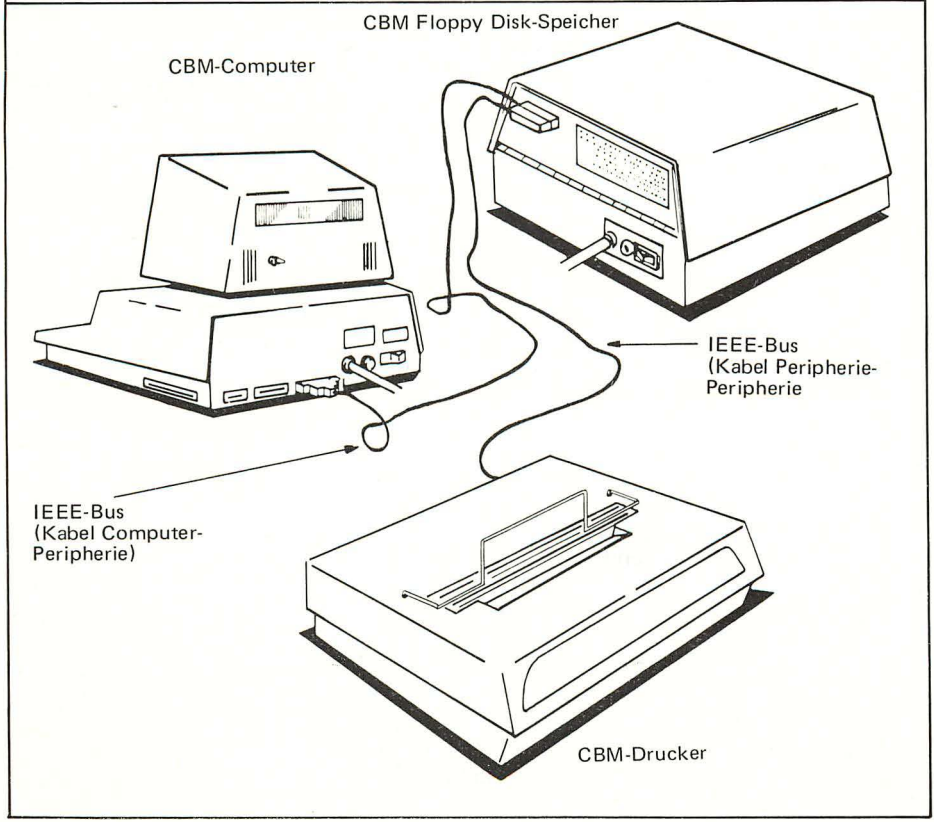
Figur 1-1. Typische Zusammenstellung eines Computer-Systems

Zum Aufbau derartiger Computersysteme weist jeder CBM-Computer auf seiner Rückseite verschiedene Anschlußmöglichkeiten auf (Figur 1-2).

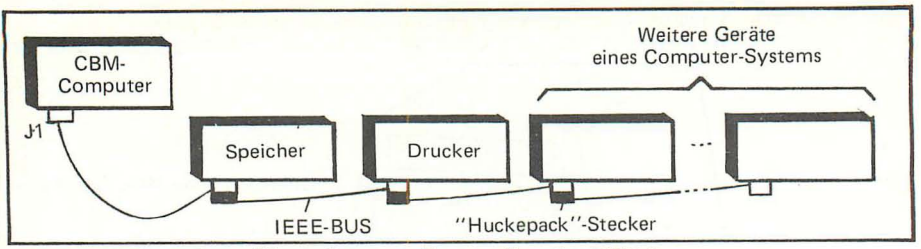
CBM-Drucker und CBM Floppy Disk-Speicher werden über eine gemeinsam benutzte Kabelverbindung, den IEEE 488-Bus, mit dem CBM-Computer verbunden (Anschluß J1 für den IEEE 488-Bus). Der IEEE-Bus ist Teil eines standardisierten Systems zum Datenaustausch zwischen Computern und programmsteuerbaren Geräten; kennzeichnend für dieses Bus-System ist, daß nicht jedes Gerät ein eigenes Verbindungskabel zum Computer benötigt. Zum Aufbau eines Computer-Systems ist lediglich eine durchgehende Kabelverbindung, der Bus, erforderlich, die der Reihe nach an allen Geräten vorbeiführt, die mit dem Computer verbunden werden sollen (Figur 1-3, Figur 1 - 4).

Figur 1-2. Anschlußmöglichkeiten für die Komponenten eines Computer-Systems





Figur 1-3. Anschluß von Speicher und Drucker über den IEEE-Bus



Figur 1-4. Aufbau eines größeren Computer-Systems über den IEEE-Bus

Für Geräte, deren Datenverkehr nicht dem IEEE-Standard folgt, steht an Stecker J2 ein vom Benutzer frei programmierbarer Computer-Anschluß zur Verfügung. Weder dieser Anschluß noch der Anschluß für den IEEE-Bus werden im vorliegenden Buch beschrieben; sie finden jedoch weiterführende Literatur im Anhang E.

Dieses Buch beschreibt Eigenschaft und Anwendung von CBM-Geräten wie Computer, Speicher und Drucker sowie die Möglichkeiten und den praktischen Einsatz eines Computer-Systems, das aus diesen Geräten besteht.

Es folgt eine Vorstellung der CBM-Computer, CBM-Speicher und CBM-Drucker.

CBM-COMPUTER

Dieses Buch beschreibt folgende CBM-Computer:

CBM 2001/8K

CBM 3008/3016/3032

CBM 4016/4032

CBM 8032

Nach einer kurzen Charakterisierung der einzelnen CBM-Computer folgen ausführliche Beschreibungen in den Abschnitten Technische Einzelheiten, Bildschirme und Tastaturen, Zeichensätze, Editionsmöglichkeiten.

CBM 8032

Der CBM 8032-Computer ist in Figur 1-5 dargestellt. Dieser Computer aus der 8001-Serie steht auf dem vorläufig letzten Entwicklungsstand der CBM-Computer-Serien. Seine besonderen Merkmale sind der große Bildschirm mit 25 Zeilen je 80 Zeichen, eine große Schreibmaschinentastatur für kommerziellen Einsatz und die Ausrüstung mit der neuesten Version der Programmiersprache BASIC mit der Bezeichnung CBM BASIC 4.0. Mit CBM BASIC 4.0 sind alle Speichermöglichkeiten ansprechbar, die das Betriebssystem DOS 2 für die neueren Floppy Disk-Speicher wie CBM 8050 und CBM 4040 ermöglicht. Gegenüber früheren Serien wurde die Edition von Bildschirmtext durch einige neue Tasten und Funktionen erweitert. Der CBM 8032 ist mit einem 32K-Byte-Speicher zur freien Belegung durch den Anwender ausgerüstet. Eine typische System-Konfiguration entsteht durch Anschluß des Floppy Disk-Speichers CBM 8050 und eines Druckers wie CBM 8024.

Figur 1-5. CBM 8001-Serie

CBM 4001-SERIE

Ein Modell der CBM 4001-Serie ist in Figur 1-6 dargestellt. Diese Computer sind unter der Bezeichnung CBM 4016 mit einem 16K-Byte-Speicher und als CBM 4032 mit einem 32K-Byte-Speicher auf den Markt gekommen. Kennzeichnend für diese Computer-Serie sind ein Bildschirm mit 25 Zeilen je 40 Zeichen, d.h. halb so breit wie der Bildschirm der CBM 8001-Computer. Für diese Computer stehen zwei Tastaturen zur Auswahl: Eine kaufmännische Tastatur wie beim CBM 8032 und eine graphische Tastatur, wie sie das Modell in Figur 1-6 zeigt, bei der der verfügbare graphische Zeichensatz vorne auf den Tasten sichtbar gemacht ist. Programmiersprache dieser Computer ist CBM BASIC 4.0, mit der neuere Floppy Disk-Speicher im Betriebssystem DOS 2 ansprechbar sind, wie z.B. der CBM 4040. Mit Computern der CBM 4001-Serie entsteht ein typisches Computer-System durch Anschluß des externen Bandgeräts CBM C2N, bei der Notwendigkeit für schnelleren Datenaustausch, des Floppy Disk-Speichers CBM 4040 sowie eines Druckers wie CBM 3022.

CBM 3001-SERIE

Ein Modell dieser Serie ist in Figur 1-7 dargestellt. Diese Computer sind mit 3 Speicherausrüstungen (8K Byte beim CBM 3008, 16K Byte beim CBM 3016 und 32K Byte beim CBM 3032) sowie mit 2 verschiedenen Tastaturen, einer kaufmännischen Tastatur und einer Tastatur mit Graphikzeichen, auf den Markt gekommen. Figur 1-6 gibt ein Beispiel für die Tastatur mit Graphikzeichen. Computer dieser Serie

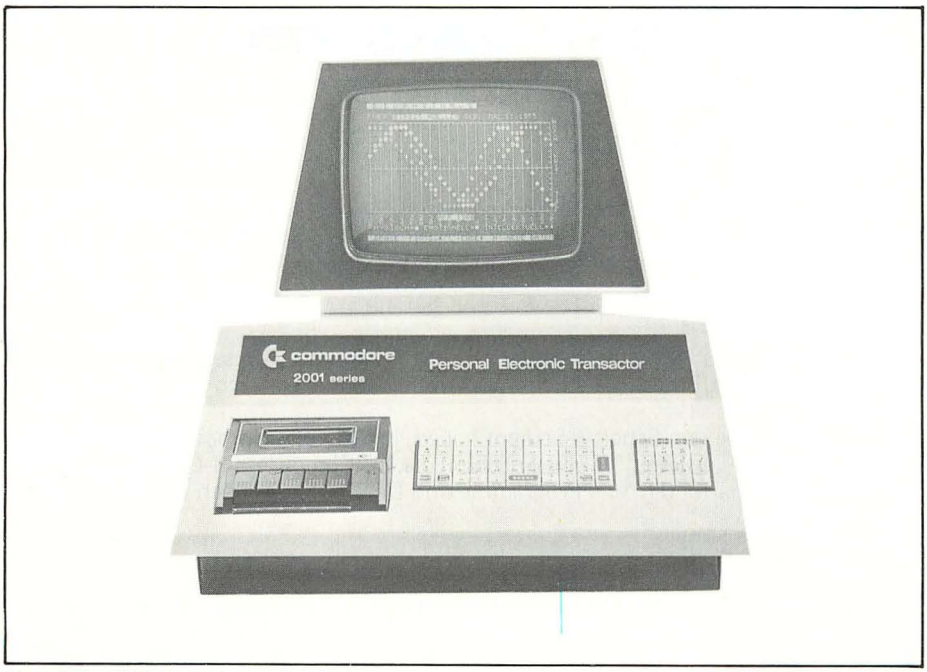
sind mit dem kleineren Bildschirm für 25 Zeilen je 40 Zeichen ausgerüstet und arbeiten in einer Version der BASIC-Programmiersprache, für die in diesem Buch die Notation CBM BASIC < 3.0 verwendet wird. Computer-Systeme dieser Serie entstehen typisch durch Anschluß eines Bandgeräts CBM C2N oder eines Floppy Disk-Speichers CBM 3040 sowie eines Druckers CBM 3022.

Figur 1-6. CBM 4001-Serie

CBM 2001-SERIE

Figur 1-8 zeigt ein Modell dieser Serie. Das auffälligste Kennzeichen dieses CBM-Computers ist ein eingebautes Bandgerät, das links neben dem mehrfarbigen, kompakten Tastenfeld angeordnet ist. Der Computer ist mit einem Bildschirm für 25 Zeilen je 40 Zeichen sowie einem frei belegbaren Speicher von 8K Byte ausgerüstet. Am Stecker J3 besteht Anschlußmöglichkeit für ein weiteres, externes Bandgerät. Bei CBM 2001-Computern mit Version-3 ROM-Speichern ist außderdem der Anschluß eines Druckers und eines Floppy Disk-Speichers möglich. Die Programmiersprache dieser Computer wird im vorliegenden Buch unter CBM BASIC < 3.0 beschrieben.

Figur 1-7. CBM 3001-Serie



Figur 1-8. CBM 2001-Serie

Nach der Darstellung technischer Einzelheiten (Figur 1-9) für Inbetriebnahme der Computer und Zusammenstellen eines Computer-Systems folgen Beschreibungen typischer Eigenschaften von CBM-Computern, die eine Vorstellung von den Anwendungsmöglichkeiten dieser Computer vermitteln.

Figur 1-9. Zur Beschreibung von CBM-Computern

TECHNISCHE EINZELHEITEN

RÜCKANSICHT

Alle Schalter, Stecker, Sicherungen und Typenangaben befinden sich auf der Rückseite Ihres CBM-Computers. Figur1-10zeigt die Rückansicht eines CBM-Computers mit den Bezeichnungen für jede wichtige Komponente, deren Bedeutung im folgenden beschrieben wird.

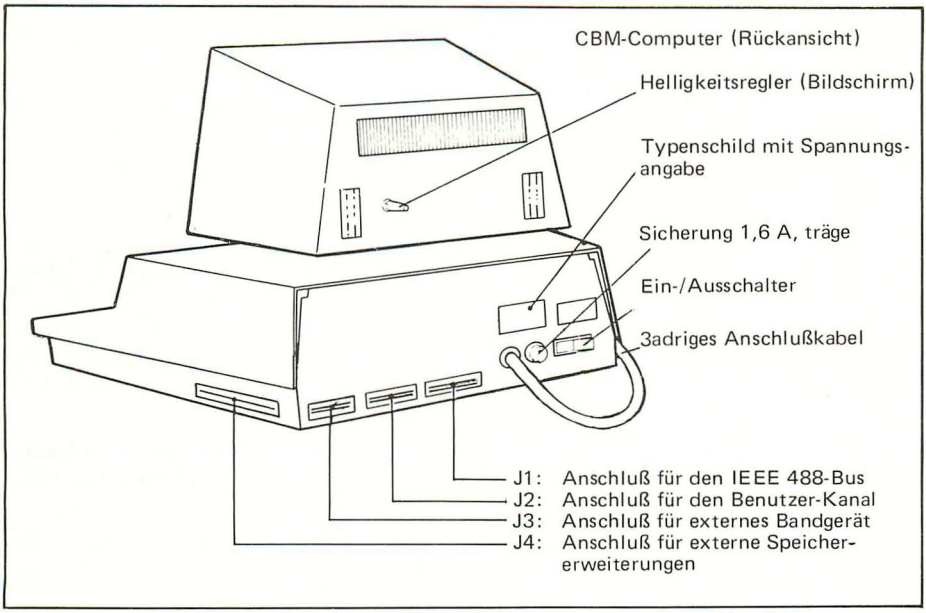
EIN/AUS-SCHALTER

Der Ein/Aus-Schalter befindet sich auf der linken Rückseite Ihres Computers; durch Druck auf die äußere Seite des Wipp-Schalters nehmen Sie den Computer in Betrieb.

Sobald Sie den Schalter bedient haben, ist Ihr CBM-Computer betriebsbereit. Beim Ausschalten verlieren Sie alle Angaben, die Ihr Computer im Lese/Schreib-Speicher gespeichert hatte, d.h. alle Programme und Daten, die Sie nach dem Einschalten eingegeben haben.

ANSCHLUSSKABEL

Über ein 3-adriges Anschlußkabel können Sie Ihren CBM-Computer an jede Haushaltssteckdose mit Schutzkontakt anschließen. Prüfen Sie zuvor die Spannungsangabe auf dem Typenschild auf der Rückseite Ihres Computers (Figur 1-10)



Figur 1-10. Technische Einzelheiten

IEEE 488-ANSCHLUSS

Der IEEE-488-Anschluß J1 befähigt Ihren CBM-Computer zum Datenaustausch mit externen Geräten. Insbesondere werden über diesen Anschluß CBM-Drucker, CBM Floppy Disk-Speicher und CBM-Bandgeräte mit Ihrem Computer verbunden. Diese Datenverbindung hat in den USA die Normbezeichnung IEEE 488, international die Normbezeichnung IEC 625 und in Deutschland die Normung DKE 66.22 (2) gefunden. Alle elektronischen Geräte, die über Datenverbindungen dieser Norm ansprechbar sind, können mit Ihrem CBM-Computer in Dialog treten.

BENUTZER-KANAL

Dieser Anschluß (Stecker J2 in Figur 1-10) kann anstelle des IEEE 488-Anschlusses für den Datenverkehr mit externen Geräten verwendet werden. Sie brauchen jedoch nichts über diesen Anschluß zu wissen. Sollten Sie ein externes Gerät haben, das über diesen Anschluß mit Ihrem Computer verbunden wird, dann enthält die Dokumentation zu diesem Gerät erforderliche Angaben für die Datenverbindung.

ANSCHLUSS FÜR BANDGERÄTE

Der Anschluß J3 in Figur 1-10 dient ausschließlich dem externen Anschluß von CBM-Bandgeräten des Typs C2N. Dieser Anschluß befindet sich auf der äußersten rechten Hinterseite Ihres Computers und ist durch seine kleinere Abmessung leicht erkennbar.

ANSCHLUSS FÜR SPEICHERERWEITERUNGEN

Anschluß J4 auf der rechten hinteren Seite Ihres CBM-Computers dient dem Anschluß externer Lese/Schreib-Speicher, über den durch Anschluß externer Speicher die Speicherkapazität Ihres CBM-Computers erweitert werden kann. Auch dieser Stecker ist für Ihr Arbeiten mit dem Computer zunächst ohne Bedeutung.

HELLIGKEITSREGLER

Mit diesem Drehknopf können Sie die Helligkeit der Bildschirmdarstellung verändern. Drehen Sie mit Blick auf den Bildschirm den Drehknopf nach links, um das Bild dunkler werden zu lassen, und nach rechts, um die Darstellung heller werden zu lassen. Beachten Sie die Änderung der Kontrastschärfe während der Helligkeitsregelung.

INBETRIEBNAHME

Folgen Sie zur Inbetriebnahme Ihres CBM-Computers folgenden Schritten:

1. Schließen Sie den Computer über das Anschlußkabel auf der Rückseite an eine Steckdose mit Schutzkontakt an. Versuchen Sie niemals den Anschluß des Computers an eine 2polige Steckdose ohne Schutzkontakt. Bei fehlender Schutzerde können Sie bei Berühren des Computer-Gehäuses einen elektrischen Schlag erhalten.
2. Schalten Sie den Computer mit dem Wipp-Schalter auf der linken Rückseite ein. Drücken Sie zum Einschalten auf die äußere Seite des Wipp-Schalters.
3. Warten Sie ab, bis eine Meldung von etwa folgender Form auf dem Bildschirm erscheint, möglicherweise angekündigt durch einen Signalton:

```
*** COMMODORE BASIC 4.0 ***

 31743 BYTES FREE

READY.
```

Die 4 Zeilen der Bildschirmdarstellung haben folgende Bedeutung:

```
*** COMMODORE BASIC 4.0 ***
```

Diese Zeile gibt an, daß die Programmiersprache BASIC in Ihrem Computer aktiviert wurde.

```
31743 BYTES FREE
```

Diese Zeile zeigt, wieviel Speicherplatz für Sie frei ist:

- 7167 BYTES bei einem 8K-Computer
- 15359 BYTES bei einem 16K-Computer
- 31743 BYTES bei einem 32K-Computer

```
READY
```

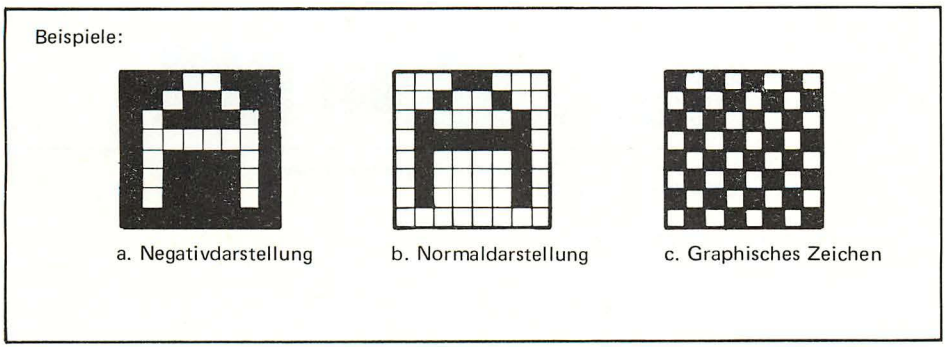
Diese Meldung zeigt an, daß der Computer für Ihre Eingaben bereit ist. Der blinkende Kursor gibt die Stelle an, an der Ihre Tasteneingabe dargestellt wird.

Wenn Sie keine Darstellung wie im Beispiel oben erhalten, schalten Sie den Computer wieder aus, warten einige Sekunden und schalten ihn dann wieder ein. Es kann vorkommen, daß sich der Bildschirm für etwa 1 Sekunde mit willkürlichen Symbolen füllt. Das ist normal; ignorieren Sie diese Darstellungen, die manchmal erscheinen, wenn ein CBM-Computer aus und nach etwa 10 Sekunden wieder eingeschaltet wurde.

BILDSCHIRME

Der Bildschirm ist ähnlich dem eines Fernsehgerätes, jedoch mit höherer Auflösung, wodurch Sie kleine graphische Symbole und Zeichen mit schärferem Kontrast sehen können. Computer der Serie CBM 8001 sind mit Bildschirmen für 2000 Zeichenpositionen ausgerüstet, angeordnet in 25 Zeilen zu je 80 Zeichen. Alle übrigen CBM-Serien sind mit Bildschirmen für 1000 Zeichenpositionen, angeordnet in 25 Zeilen zu je 40 Zeichen, ausgerüstet. Jeder Zeichenposition entspricht ein Speicherplatz im Bildschirmspeicher; jedes Zeichen wird vor seiner Darstellung auf dem Bildschirm zunächst im Bildschirmspeicher abgelegt.

Die Bildschirme verwenden ein Raster von 8 x 8 Punkten; alle Zeichen werden durch Hellsteuerung von Punkten in diesem Raster zur Darstellung gebracht (Figur 1-11). Ebenso sind Negativdarstellungen aller Zeichen auf dem Bildschirm möglich; hierbei werden alle Rasterpunkte, die nicht zum Zeichen gehören, hell gesteuert (s. Beispiel a in Figur 1-11).



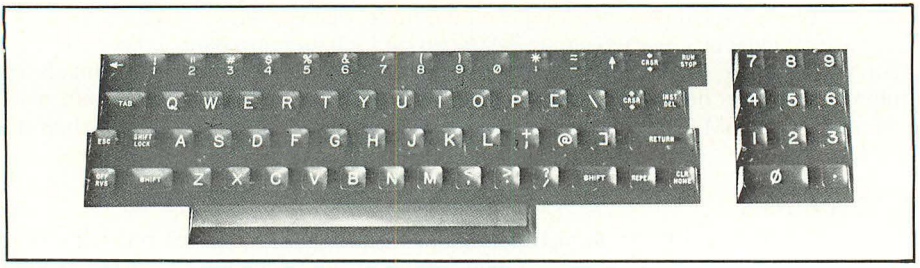
Figur 1-11. Bildschirmdarstellungen im 8 x 8-Raster

TASTATUREN, ZEICHENSÄTZE UND EDITIONSMÖGLICHKEITEN

TASTATUREN

Über das Tastenfeld Ihres CBM-Computers werden sowohl Programme als auch Daten, die von Programmen benötigt werden, eingegeben. Der Aufbau des Tastenfelds hängt von dem Modell des CBM-Computers ab, das Sie gewählt haben. Es gibt folgende drei Varianten des Tastenfeldes:

1. **Große Schreibmaschinen-Tastatur**
   Diese Tastatur wurde für den kommerziellen Einsatz von CBM-Computern entwickelt und ist für Computer der Serien CBM 3001, 4001 und 8001 erhältlich. Abbildungen dieser Tastatur finden Sie in Figur 1-5 und Figur 1-12.



Figur 1-12. Große Schreibmaschinen-Tastatur (CBM 8032)

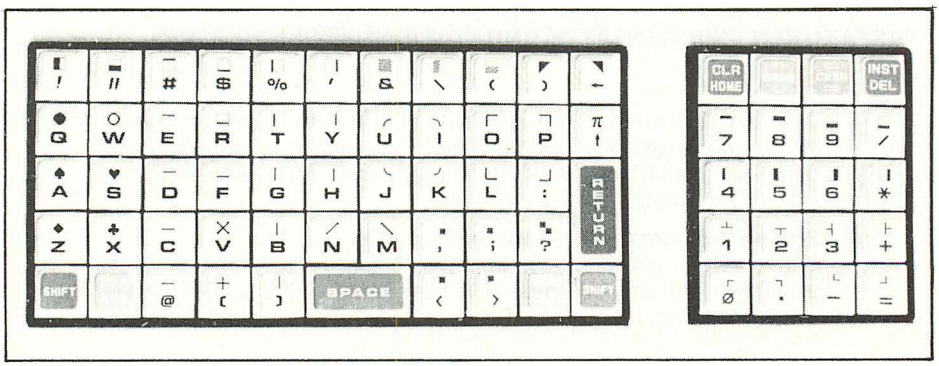
2. Große Schreibmaschinen-Tastatur mit Graphik-Symbolen

Diese Tastatur ist bei Computern der Serien CBM 3001 und 4001 erhältlich und wurde für technisch-wissenschaftliche Einsätze entwickelt. Der Unterschied zur oben genannten Tastatur besteht darin, daß die auf CBM-Computern verfügbaren graphischen Symbole und Sonderzeichen auf der Vorderseite der Tasten abgebildet sind. Figur 1-13 zeigt ein Beispiel dieser Tastatur.

Figur 1-13. Große Schreibmaschinen-Tastatur mit Graphiksymbolen (Serien CBM 3001, CBM 4001)

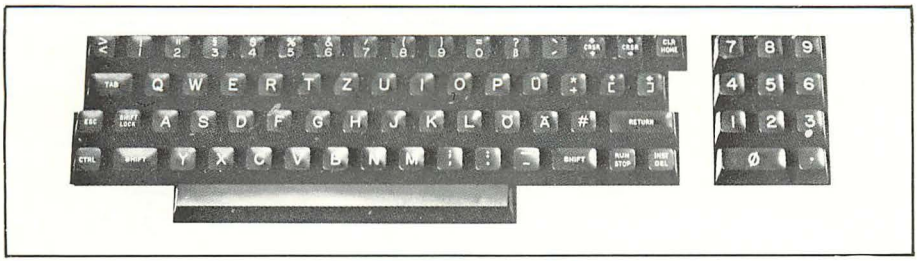
3. Kleine Tastatur

Diese Tastatur ist nur auf CBM 2001-Computern zu finden, bei denen links neben der Tastatur ein Bandgerät angeordnet ist. Auffallend für diese Tastatur sind ihre mehrfarbigen Tasten. Figur 1-14 zeigt einen Blick auf diese Tastatur.



Figur 1-14. Kleine Tastatur der CBM 2001-Computer

Für Computer der Serie CBM 8001 wurde außerdem eine Tastatur für den deutschen Sprachraum entwickelt, die neben einer veränderten Anordnung der Tasten auch die Buchstaben Ä, Ö, Ü und ß enthält (Figur 1-15).



Figur 1-15. Tastatur für den deutschen Sprachraum (mit Ä, Ö, Ü, ß)

Mit wenigen Ausnahmen sind die gleichen Tasten sowohl auf der kleinen Tastatur als auch auf der Schreibmaschinen-Tastatur vorhanden. Die Tasten auf dem Tastenfeld von CBM-Computern bilden folgende Gruppen: Alphabetische Tasten, numerische Tasten, Tasten für Sonderzeichen, graphische Tasten, Funktionstasten und Tasten zur Kontrolle des Kursors.

ALPHABETISCHE TASTEN

Die alphabetischen Tasten tragen die 26 Buchstaben des Alphabets und, auf der Tastatur für CBM 8032-Computer, auch die Umlaute Ä, Ö, Ü und ß. Auf allen CBM-Modellen stehen Groß- und Kleinbuchstaben zur Verfügung.

Auf der Tastatur Ihres CBM-Computers stehen Ihnen zwei Zeichensätze zur Verfügung: der Standard- und der Alternativ-Zeichensatz (siehe auch Anhang A, Tabelle A-4).

Aus der Bildschirmdarstellung erkennen Sie, welcher der Zeichensätze augenblicklich aktiv ist: Im Standard-Zeichensatz stellen die Tasten A - Z Großbuchstaben und, bei gedrückter SHIFT-Taste, graphische Symbole dar (Tabelle 1-1). Im Alternativ-Zeichensatz werden dagegen Kleinbuchstaben und, wie bei einer Schreibmaschine, bei gedrückter SHIFT-Taste Großbuchstaben dargestellt.

Mit folgenden Befehlseingaben, jeweils mit der Taste RETURN abgeschlossen, können Sie zwischen beiden Zeichensätzen umschalten:

Standard-Zeichensatz:

```
POKE 59468,12
```

oder

```
?CHR$(142)  bei CBM 8032-Computern
```

Alternativ-Zeichensatz:

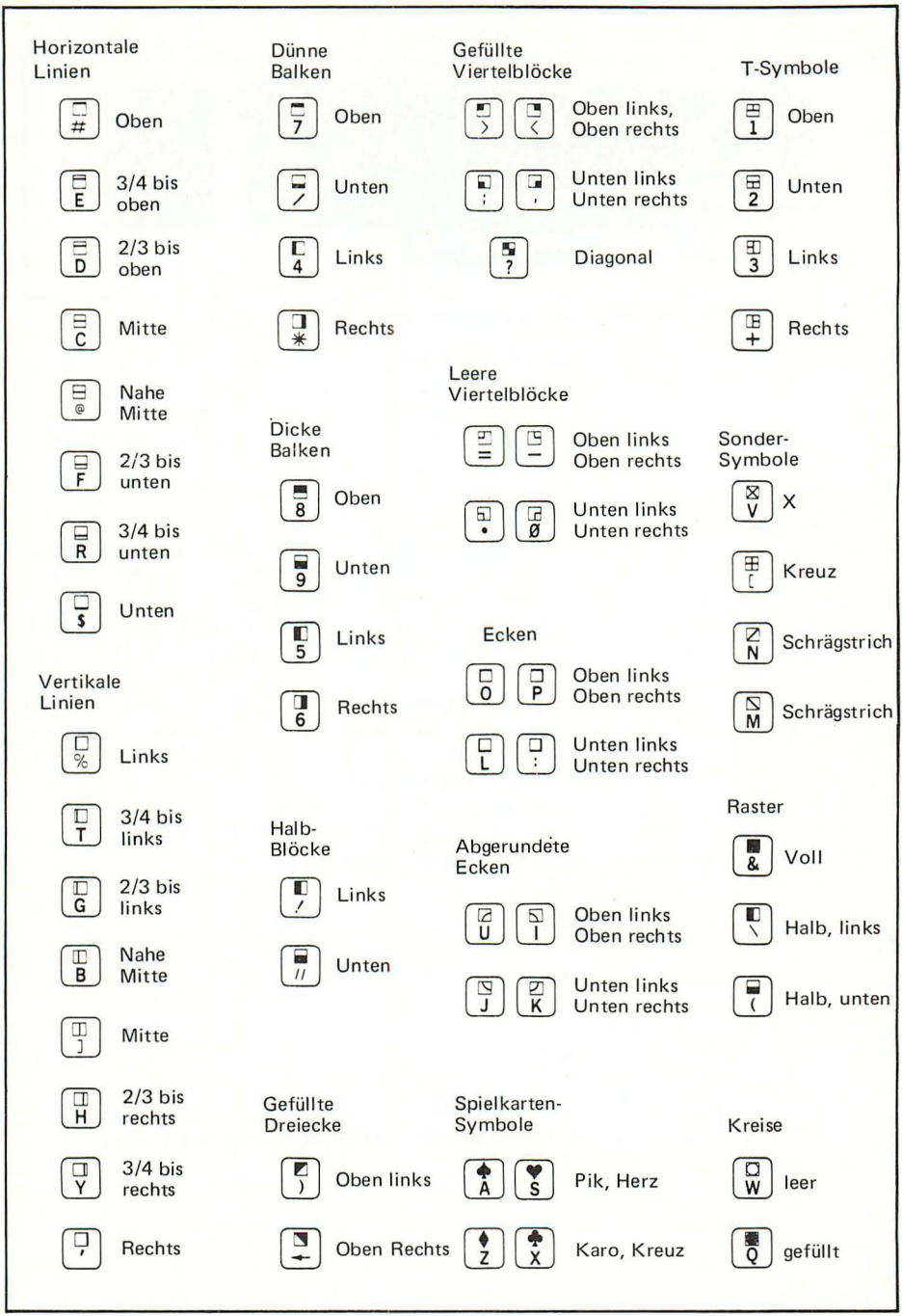
```
POKE 59468,14
```

oder

```
?CHR$(14)  bei CBM 8032 Computern
```

Sonderzeichen und die Zifferntasten 0 - 9 bleiben von Umschaltungen zwischen den Zeichensätzen unberührt.

Tabelle 1-1 . Graphische Symbole auf CBM Computern.



NUMERISCHE TASTEN

Die numerischen Tasten tragen die Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Alle Ziffern können außerdem über ein kleines, gesondertes Tastenfeld auf der rechten Seite der Tastatur eingegeben werden, auf dem die Taste mit der Ziffer 5 durch eine kleine Erhöhung gekennzeichnet ist.

TASTEN FÜR SONDERZEICHEN

Zu den Sonderzeichen gehören die Zeichen der Zeichensetzung, die Operator-Zeichen der vier Grundrechnungsarten, das Dollarzeichen, das auf deutschen Tastaturen unbekannte @-Zeichen und andere.

Einige dieser Sonderzeichen haben besondere Bedeutung in der Programmiersprache BASIC. Nähere Angaben finden Sie in Kapitel 4.

GRAPHISCHE TASTEN

Das Tastenfeld von CBM-Computern enthält 62 graphische Symbole, die im "Standard-Zeichensatz" (s. unten) nach Betätigung der Umschalttaste ausgegeben werden. Mit derart vielen graphischen Symbolen eignen sich CBM-Computer zum Entwurf komplexer Bildschirmzeichnungen.

Die graphischen Symbole sind in Tabelle 1-1 zusammengestellt; jedem Zeichen wurde eine Beschreibung beigefügt und ähnliche Zeichen zu Gruppen zusammengefaßt. Beachten Sie, daß das Viereck, in dem die graphischen Symbole dargestellt sind, nicht Teil des graphischen Symbols ist und auch nicht auf dem Bildschirm erscheint. Aufgabe des Vierecks ist, die Lage des graphischen Symbols innerhalb des 8x8-Punktrasters sichtbar zu machen (s. Abschnitt Bildschirme).

Auf der Tastatur Ihres CBM-Computers stehen Ihnen zwei Zeichensätze zur Verfügung: der Standard- und der Alternativ-Zeichensatz (siehe auch Anhang A, Tabelle A-4).

Aus der Bildschirmdarstellung erkennen Sie, welcher der Zeichensätze augenblicklich aktiv ist: Im Standard-Zeichensatz stellen die Tasten A - Z Großbuchstaben und, bei gedrückter SHIFT-Taste, graphische Symbole dar (Tabelle 1-1). Im Alternativ-Zeichensatz werden dagegen Kleinbuchstaben und, wie bei einer Schreibmaschine, bei gedrückter SHIFT-Taste Großbuchstaben dargestellt.

Mit folgenden Befehlseingaben, jeweils mit der Taste RETURN abgeschlossen, können Sie zwischen beiden Zeichensätzen umschalten:

Standard-Zeichensatz:

```
POKE 59468,12
```

oder

```
?CHR$(142)  bei CBM 8032-Computern
```

Alternativ-Zeichensatz:

```
POKE 59468,14
```

oder

```
?CHR$(14)   bei CBM 8033-Computern
```

Sonderzeichen und die Zifferntasten 0 - 9 bleiben von Umschaltungen zwischen den Zeichensätzen unberührt.

FUNKTIONSTASTEN

Im folgenden werden alle Funktionstasten beschrieben, die Sie auf der Tastatur Ihres CBM-Computers vorfinden. Wir haben jedoch die Beschreibung von Funktionstasten, mit denen Textdarstellungen auf dem Bildschirm verändert werden können, im Abschnitt "Editionsmöglichkeiten" zusammengefaßt.

SHIFTDie SHIFT-Taste entspricht der Umschalttaste bei Schreibmaschinen. Fast jede Taste auf der Tastatur stellt ein anderes Zeichen dar, wenn sie zugleich mit der SHIFT-Taste betätigt wird. Im "Standard-Zeichensatz" (s. unten) bewirkt die SHIFT-Taste eine Umschaltung von Großbuchstaben auf graphische Symbole. Im "Alternativ-Zeichensatz" arbeitet die Tastatur wie bei einer Schreibmaschine: Bei gleichzeitiger Betätigung der SHIFT-Taste und einer Taste des Alphabets werden Großbuchstaben ausgegeben.

Auf jeder Tastatur befinden sich zwei SHIFT-Tasten: auf der linken und auf der rechten unteren Tastenreihe.

SHIFT LOCK. Die Taste SHIFT LOCK entspricht dem Umschaltfeststeller auf einer Schreibmaschine: Diese Taste rastet nach Niederdrücken über den spürbaren Widerstand hinaus ein und wirkt wie eine dauernd gedrückt gehaltene SHIFT-Taste. Diese Einstellung wird durch erneutes Drücken der gleichen Taste wieder aufgehoben.

RUN/STOP. Die Taste RUN/STOP vereinigt zwei Funktionen: Die Funktion RUN wird durch gleichzeitiges Drücken der Taste SHIFT angesprochen, STOP ohne Drücken der Taste SHIFT. STOP unterbricht jeden Programmablauf und stellt dem Benutzer das Tastenfeld und den Bildschirm für Eingaben zur Verfügung. Machen Sie eine erste Erfahrung mit der STOP-Taste, indem Sie das folgende einzeilige Programm eingeben, ohne es im Augenblick verstehen zu wollen. Geben Sie die schattierte Zeile ein. Nach Drücken der Taste RETURN wird auf dem Bildschirm eine vertikale Spalte von Zahlen dargestellt, wie folgendes Beispiel zeigt:

```
FOR I=1 TO 100:?I:NEXT I
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10 ←—— Taste STOP gedrückt

BREAK
READY.
```

Wenn Sie jetzt die Taste STOP drücken, erstarrt die Bildschirmdarstellung, da der Programmablauf unterbrochen wurde.

Die STOP-Taste bleibt ohne Wirkung, wenn es nichts zu unterbrechen gibt, d.h. wenn im CBM-Computer kein Programm abläuft.

Durch Betätigen der Tasten SHIFT und RUN wird die Funktion RUN aktiviert: hierbei wird der Ablauf eines gespeicherten Programms ausgelöst, bzw., falls sich das Programm noch in einem Bandgerät oder einem Floppy Disk-Speicher befindet, zuerst das Programm in den Computer geladen und anschließend zum Ablauf gebracht.

Zu den weiteren Funktionstasten, die im Abschnitt "Editionsmöglichkeiten" besprochen werden, gehören die Tasten RETURN, OFF, RVS, CRSR↕, CRSR↔, CLR/HOME, INST/DEL, ESC, TAB und REPEAT.

Es folgen Darstellungen der Tastaturen, auf denen die genannten Tastengruppen (alphabetische Tasten, numerische Tasten, graphische Tasten usw.) durch Schattierung sichtbar gemacht wurden.

GROSSE SCHREIBMASCHINEN-TASTATUR

Siehe Figur 1-12 für eine Abbildung dieser Tastatur.

Alphabetische Tasten

Im Alternativ-Zeichensatz werden mit den schattierten Tasten Kleinbuchstaben und, bei gleichzeitiger Betätigung der Umschalttaste SHIFT, Großbuchstaben dargestellt. Im Standard-Zeichensatz stellen die Tasten Großbuchstaben, und, bei gleichzeitiger Betätigung der SHIFT-Taste, graphische Symbole dar.

Numerische Tasten

Numerische Tasten treten zweimal auf: In der oberen Tastenreihe der Schreibmaschinen-Tastatur und auf dem kleinen Tastenfeld rechts daneben. Die Zifferntasten bleiben unberührt von Umschaltungen zwischen Alternativ- und Standard-Zeichensatz. Für Eingaben in Blindschrift ist Taste "5" auf dem kleinen Tastenfeld mit einem kleinen Höcker in der Mitte versehen.

Tasten für Sonderzeichen

Die Sonderzeichen sind in der Darstellung durch Schattierung gekennzeichnet. Befinden sich zwei Zeichendarstellungen auf einer Taste, dann wird jeweils das obere Zeichen bei gleichzeitiger Betätigung der SHIFT-Taste ausgegeben.

Graphische Tasten

Graphische Symbole sind zwar auf den Tasten der großen Schreimaschinen-Tastatur nicht dargestellt, aber verfügbar. Graphische Symbole werden im Standard-Zeichensatz bei gleichzeitigem Betätigen der SHIFT-Taste ausgegeben.

Funktionstasten

In der Darstellung sind alle Funktionstasten durch Schattierung gekennzeichnet. Die Tasten TAB, ESC und REPEAT sind nur auf den Tastaturen der Computer CBM 8032 vorhanden. Im Abschnitt "Editionsmöglichkeiten" finden Sie die Beschreibung dieser Tasten.

Kontrolltasten des Kursors

Die Funktion dieser Tasten wird im folgenden Abschnitt "Editionsmöglichkeiten" beschrieben.

SCHREIBMASCHINENTASTATUR MIT GRAPHIKSYMBOLEN

Diese Tastatur ist bei Computern der Serien CBM 3001 und CBM 4001 anzutreffen. Kennzeichen ist die Darstellung der graphischen Symbole auf der Vorderseite der Tasten (Figur 1-13).

Alphabetische Tasten

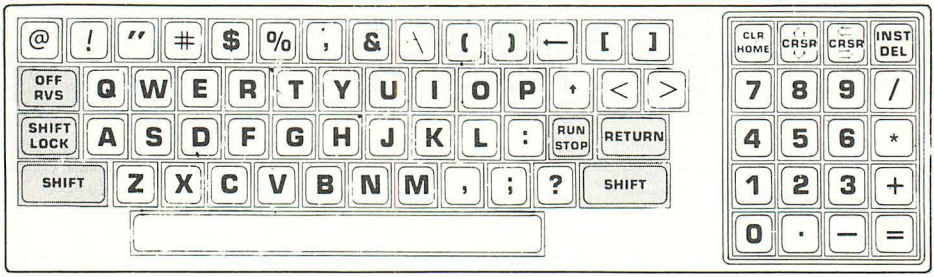
Numerische Tasten

Tasten für Sonderzeichen

Graphische Tasten

Die graphischen Symbole sind, außer bei Funktions- und Kursortasten, auf der Frontseite der Tasten dargestellt. Graphische Symbole werden nur im Standard-Zeichensatz und bei gleichzeitigem Betätigen der SHIFT-Taste ausgegeben.

Funktionstasten



Auf dieser Tastatur ist die Kreiskonstante Pi (π) enthalten. Bei gleichzeitigem Betätigen der Tasten SHIFT und ↑ wird π ausgegeben.

Pi (π) ist gleich dem Kreisumfang geteilt durch den Kreisdurchmesser. Ihr Wert ist 3.14159265, der Ihnen bei folgender Eingabe dargestellt wird:

```
?π
3.14159265

READY.
▒
```

Nach Betätigen der Taste RETURN wird der angegebene Wert dargestellt. Wenn Sie dagegen nur das Symbol π ausgeben wollen, setzen Sie es zwischen Anführungszeichen:

```
?"π"
π

READY.
▒
```

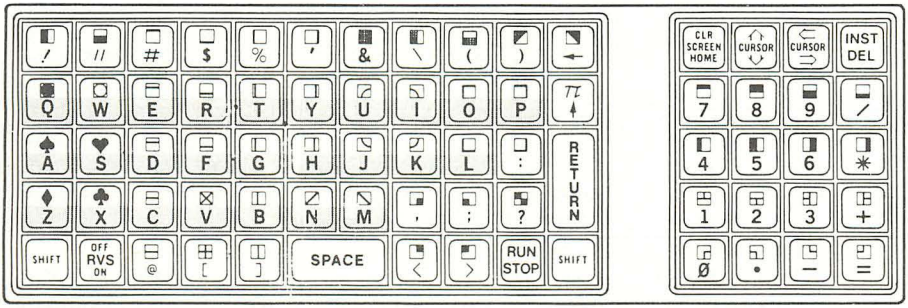
Kursor-Kontrolltasten

KLEINE TASTATUR

Computer der Serie CBM 2001 sind mit einer kompakten, mehrfarbigen Tastatur ausgestattet, über die die Zeichen des Standard-Zeichensatzes und des

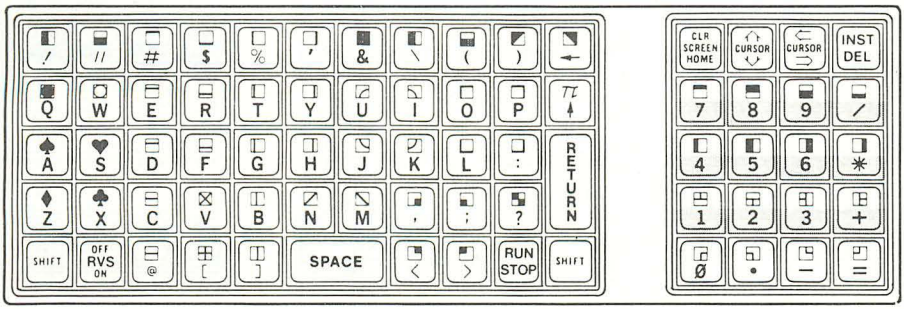
Alternativ-Zeichensatzes wie bei den anderen Tastaturen ausgegeben werden kann. (Figur 1-14).

Alphabetische Tasten



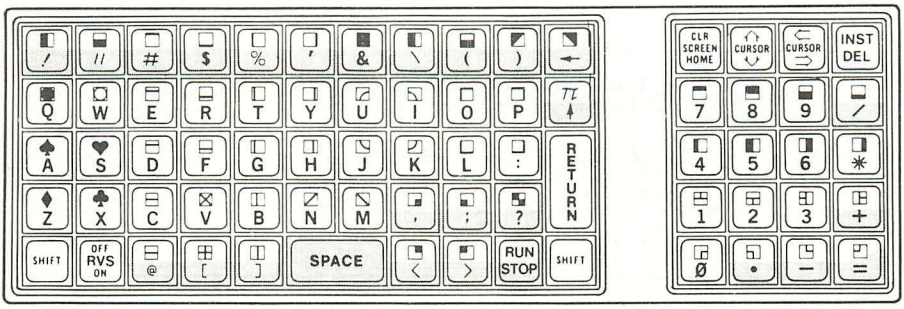
Buchstaben des Alphabets sind auf den silberfarbigen Tasten angeordnet.

Numerische Tasten



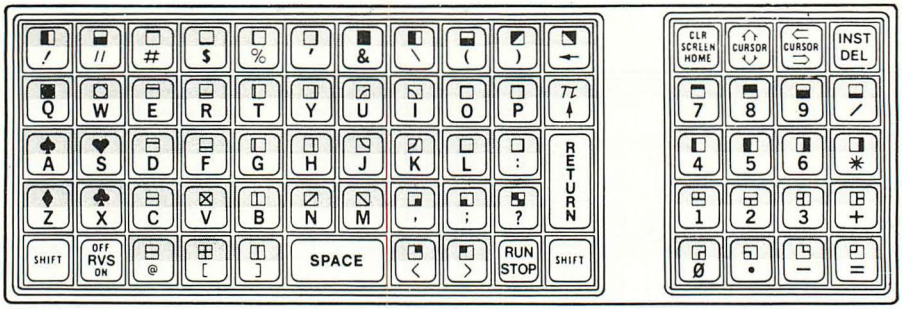
Ziffern befinden sich auf den silberfarbigen Tasten des kleinen Tastenfeldes und werden ohne Betätigung der SHIFT-Taste ausgegeben.

Tasten für Sonderzeichen



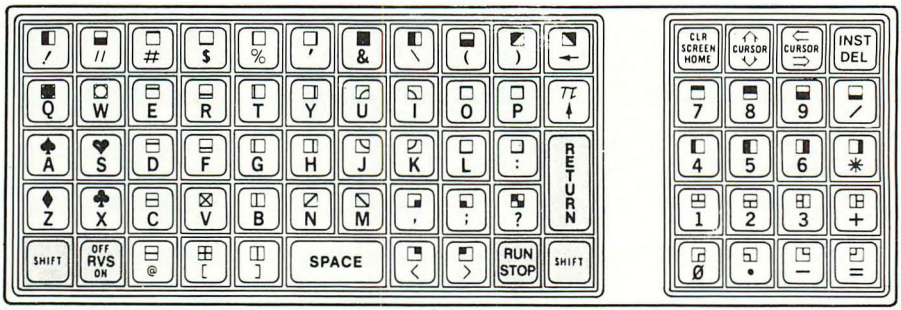
Die Tasten für Sonderzeichen sind leicht blau gefärbt.

Graphische Tasten



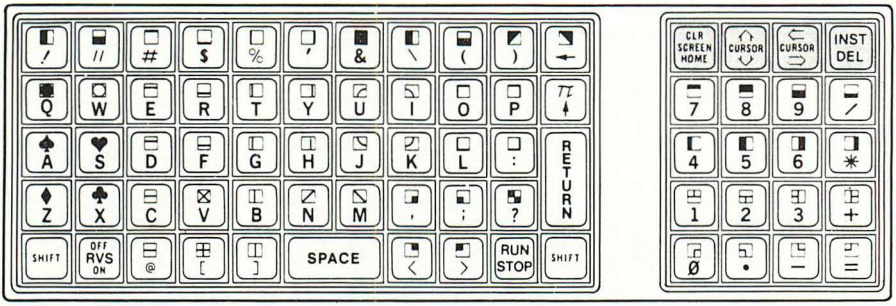
Graphische Symbole sind auf allen Tasten, außer den Funktions- und Kursortasten angegeben und werden im Standard-Zeichensatz bei gleichzeitiger Betätigung der SHIFT-Taste ausgegeben.

Funktionstasten



Auf dieser Tastatur fehlt die Taste SHIFT/LOCK. Jedoch ist eine Taste für die Kreiskonstante Pi (π) vorhanden.

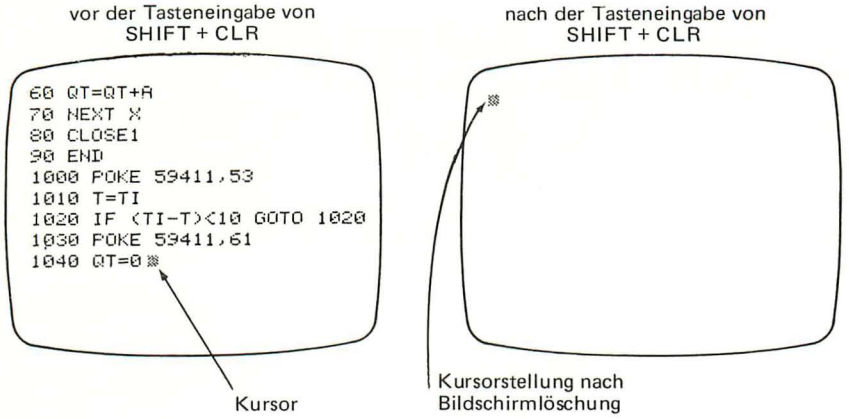
Kursor-Kontrolltasten



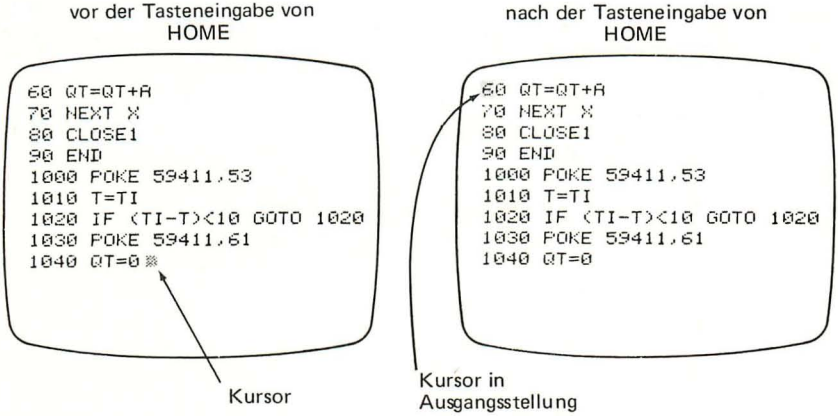
EDITIONSMÖGLICHKEITEN

Die folgenden Beschreibungen von Tastenfunktionen geben Ihnen eine Vorstellung von den Möglichkeiten der Textedition am Bildschirm von CBM-Computern.

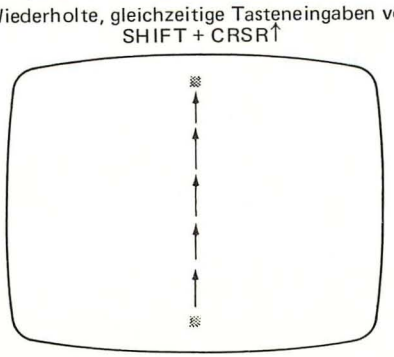
CLR/HOME. Die Tastenfunktion CLR wird durch gleichzeitiges Niederdrücken der Taste SHIFT angesprochen und bewirkt beides, eine Löschung des Bildschirms (CLR = CLEAR) und eine Rückführung des Kursors in seine Ausgangsstellung auf der linken oberen Bildschirmecke:



Dagegen bewirkt die Tastenfunktion HOME lediglich die Rückführung des Kursors in seine Ausgangsstellung (HOME = "heimschicken"):

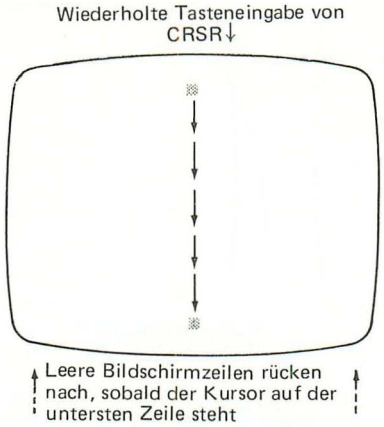


CRSR↑. Diese Tastenfunktion bewegt den Kursor auf seiner augenblicklichen Bildschirmspalte um jeweils eine Zeile nach oben:



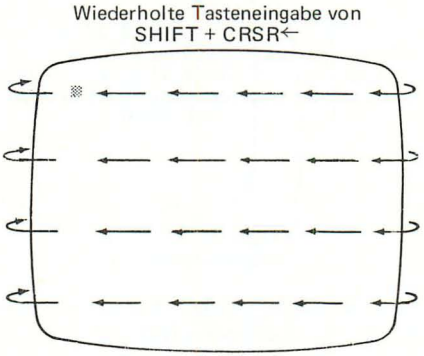
Die Eingabe erfolgt durch gleichzeitiges Drücken der Tasten SHIFT und CRSR↑. Vom Kursor überfahrere Zeichen verändern sich nicht. Ist der Kursor am oberen Bildschirmrand angekommen, dann verliert die Taste ihre Wirkung.

CRSR↓. Diese Tastenfunktion bewegt den Kursor auf seiner augenblicklichen Bildschirmspalte nach unten:



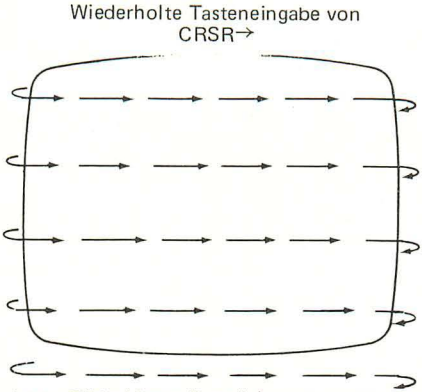
Hat der Kursor den unteren Bildschirmrand erreicht, dann bewirkt die Taste CRSR↓ ein "Aufrollen" des Bildschirms: Bei jeder Tastenbetätigung verschwindet eine Bildschirmzeile über den oberen Bildschirmrand, während eine leere Bildschirmzeile von unten nachrückt.

CRSR←. Diese Tastenfunktion bewirkt eine Linksverschiebung um eine Stelle auf der augenblicklichen Zeile:



Für diese Funktion werden die Tasten SHIFT und CRSR gleichzeitig eingegeben. Nach Erreichen des linken Bildschirmrandes springt der Kursor auf das Ende der darüberliegenden Zeile.

CRSR→. Mit dieser Tastenfunktion kann der Kursor um jeweils eine Stelle auf seiner augenblicklichen Zeile nach rechts bewegt werden:



Leere Bildschirmzeilen rücken von unten nach,
sobald der Kursor rechts unten angekommen ist.

Nach Erreichen des rechten Bildschirmrandes springt der Kursor auf den Anfang der darunterliegenden Zeile. Am unteren rechten Bildschirmende angekommen, bewirkt CRSR→ das gleiche wie CRSR↓: Der Bildschirm wird nach oben gerollt und eine leere Bildschirmzeile rückt von unten nach.

INST/DEL. Die Tastenfunktion DEL löscht ein unmittelbar links vom Kursor stehendes Zeichen, wobei der Kursor samt einem rechts von ihm stehenden Text um eine Stelle nach links wandert:

```
ICH WILL NICHT MEHR
ICH WILL NICH MEHR
ICH WILL NIC MEHR
ICH WILL NI MEHR
ICH WILL N MEHR
ICH WILL  MEHR
ICH WILL MEHR
```

Textlöschung mit
der Taste DEL

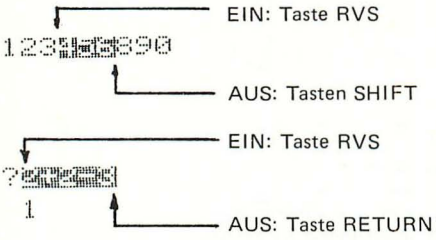
Zusammen mit der Taste SHIFT bewirkt die Tastenfunktion INST (= INSERT) das Öffnen eines Textes an der augenblicklichen Kursorstellung, um Platz für eine Texteinfügung zu machen:

```
ICH WILL MEHR
ICH WILL  MEHR
ICH WILL   MEHR
ICH WILL    MEHR
ICH WILL     MEHR
ICH WILL      MEHR
ICH WILL NICHT MEHR
```

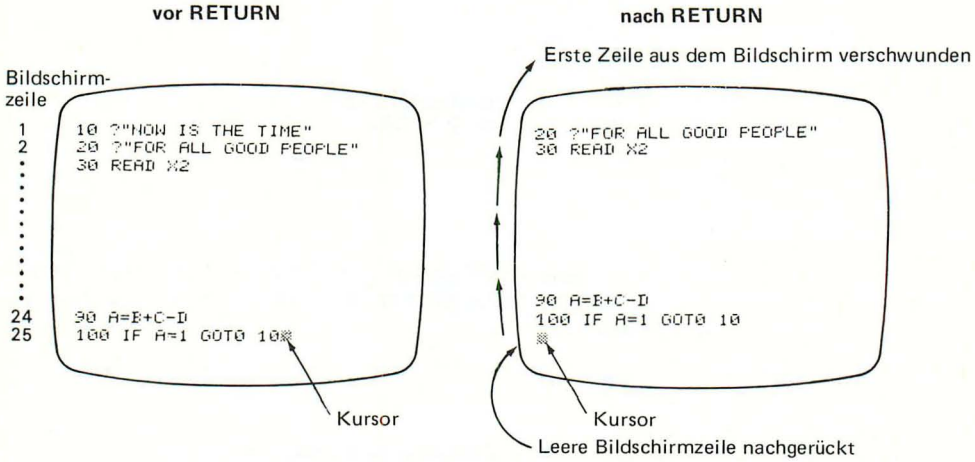
Texteinfügung mit den
Tasten SHIFT und INST

CBM-Computer behandeln Bildschirmtext als eine Folge von Zeilen mit jeweils 80 Zeichen. Texteinfügung und vorhandener Text dürfen also nicht mehr als 80 Zeichen lang werden. Ebenso beim Löschen: Sie können bis zum Zeilenanfang, aber nicht darüber hinaus Text löschen.

OFF/RVS. Diese Tastenfunktion ermöglicht die Textdarstellung in Negativschrift (s. auch Figur 1-11). Zur Umschaltung von Normalschrift auf Negativschrift drücken Sie zuvor die Taste OFF/RVS. Mit der Tasteneingabe SHIFT und OFF/RVS können Sie von Negativschrift zurückschalten auf Normalschrift. Die Negativschrift wird auch durch Betätigen der Taste RETURN unterbrochen:



RETURN. Die Tastenfunktion RETURN entspricht dem Wagenrücklauf auf der Schreibmaschine; bei Betätigung der Taste bewegt sich der Kursor auf den Anfang der folgenden Bildschirmzeile. Befindet sich der Kursor auf der letzten Bildschirmzeile, dann bewirkt die Tasteneingabe RETURN ein "Aufrollen" des Bildschirms: Die erste Bildschirmzeile verschwindet über den oberen Bildschirmrand, während von unten eine leere Bildschirmzeile nachrückt.



TAB. Mit dieser Taste können die Positionen von Tabellen gesetzt bzw. gelöscht werden und bei der Tabelleneingabe der Sprung zur nächsten Tabellenspalte ausgelöst werden (s. Kapitel 6 und Kapitel 8). Diese Taste finden Sie nur bei Computern der Serie CBM 8001.

ESC (CBM 8032). Diese Taste hat zwei Funktionen: Sie löscht die Wirkung einer INST-, DEL- und RVS-Taste; außerdem wird sie für besondere Editionsfunktionen verwendet (s. Kapitel 5).

REPEAT (CBM 8032). Jede gleichzeitig mit REPEAT betätigte Taste wird zu einer Wiederholtaste: d.h. die Tastenausgabe wird automatisch wiederholt.

CBM-SPEICHER

Die Speicherung von Programmen und Daten in Ihrem CBM-Computer hat einen Nachteil: Beim Ausschalten des Computers gehen diese Informationen verloren. Zur dauerhaften Speicherung von Programmen und großen Datenmengen wurden Speichergeräte entwickelt, die nach dem gleichen Prinzip arbeiten wie Tonbandgeräte oder Kassettenrekorder.

Es folgen Beschreibungen von CBM-Bandgeräten und CBM Floppy Disk-Speichern, deren Benutzung zur Speicherung von Programmen oder Daten ausführlich in diesem Buch dargestellt ist (siehe besonders Kapitel 6).

CBM-BANDGERÄT C2N

Dieses Bandkassettengerät ist integriert in das Gehäuse von CBM 2001-Computern (Figur 1-4) und wird an alle anderen CBM-Computer extern, d.h. über eine Kabelverbindung angeschlossen (Figur 1-16). Internes und externes Bandgerät unterscheiden sich nicht.

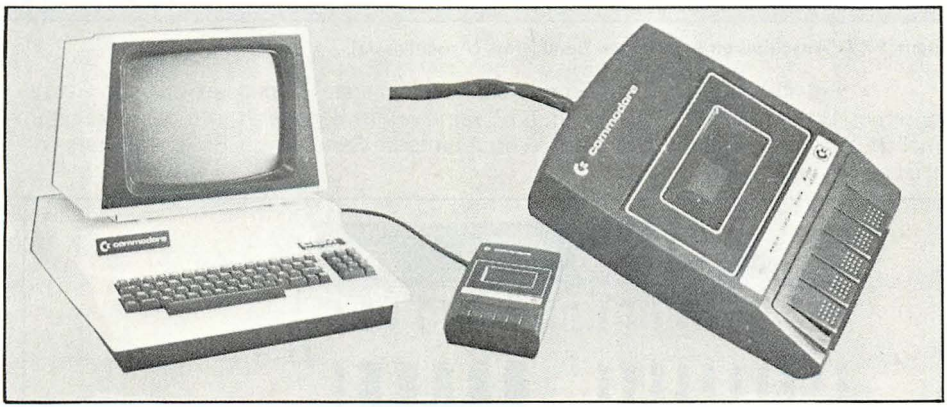


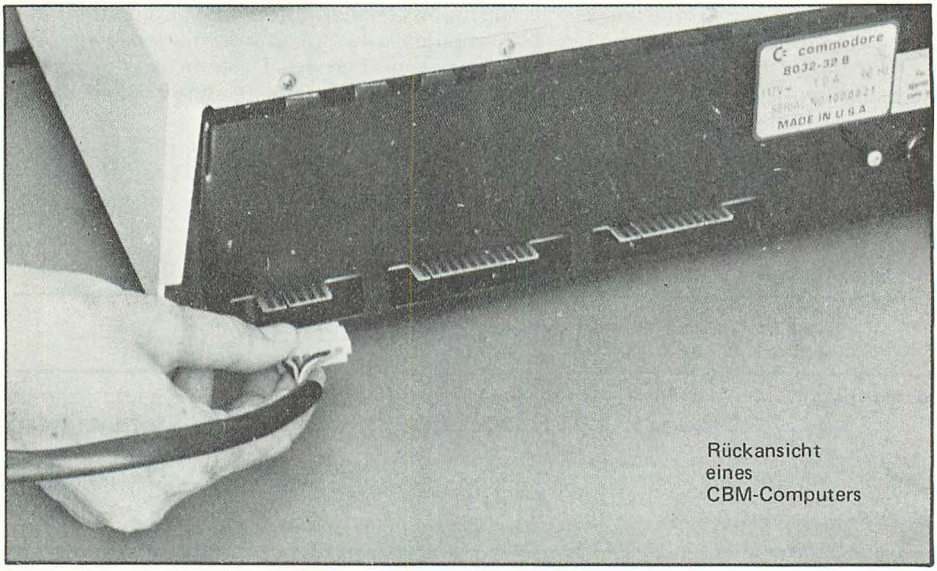
Figure 1-16. Das externe Bandgerät CBM C2N

Das Bandgerät C2N ermöglicht Ihnen die Speicherung von Programmen und Daten auf einer Kassette. Sie können dann mit Ihrem CBM-Computer diese Programme und Daten von der Kassette lesen und in den Speicher Ihres Computers zurückbringen.

EXTERNER ANSCHLUSS

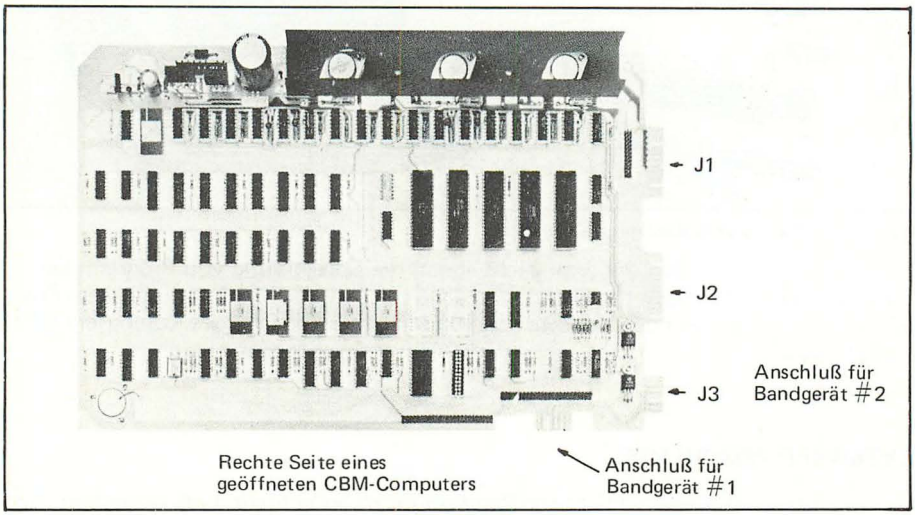
Das extern anzuschließende Bandgerät C2N ist in Figur 1-16 abgebildet. An einem CBM-Computer können zur gleichen Zeit höchstens zwei Bandgeräte betrieben

werden. Im Falle eines CBM 2001-Computers mit eingebautem Bandgerät wird ein zweites, externes Bandgerät über den Anschluß J3 betrieben (s. Figur 1-10 und Figur 1-17). Das eingebaute Bandgerät trägt die Gerätenummer 1, das externe Bandgerät die Gerätenummer 2.



Figur 1-17. Anschluß eines externen Bandgeräts (Anschluß J3)

Bei allen übrigen CBM-Computern wird ein erstes Bandgerät (mit der Gerätenummer 1) über Anschluß J3 (Figur 1-17), ein zweites Bandgerät (mit der Gerätenummer 2) über einen Anschluß auf der rechten hinteren Seite Ihres CBM-Computers betrieben (Figur 1-18).

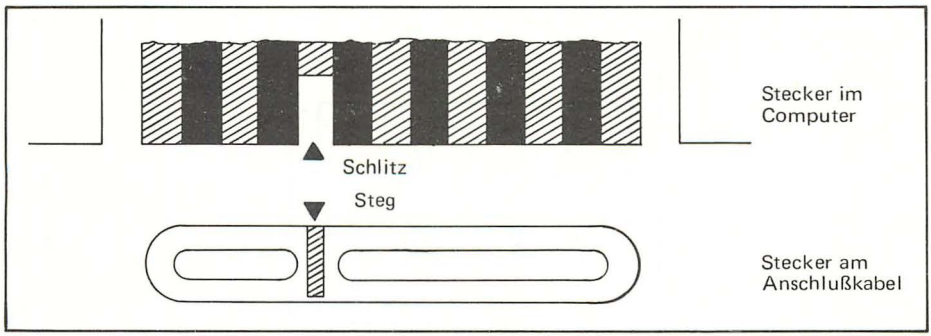


Figur 1-18. Anschluß eines zweiten, externen Bandgeräts

Alle weiteren externen Geräte wie Drucker und Floppy Disk-Speicher werden über den Anschluß J1 für den IEEE-488-Bus betrieben.

ANSCHLUSS

Beim Anschluß eines externen Bandgerätes können Sie entweder die Steckerverbindung richtig herstellen, oder einen Stecker zerbrechen. Zur Vermeidung von Anschlußfehlern wurde eine asymmetrische Steckerkonstruktion gewählt, bei der Voraussetzung für eine richtige Steckerverbindung mit dem Computer ist, daß der Stecker nicht seitenverkehrt eingeführt wurde (Figur 1-19).



Figur 1-19. Schutz gegen Anschlußfehler durch asymmetrische Steckerkonstruktion

Gehen Sie zum Anschluß eines externen Bandgerätes in folgenden Schritten vor:

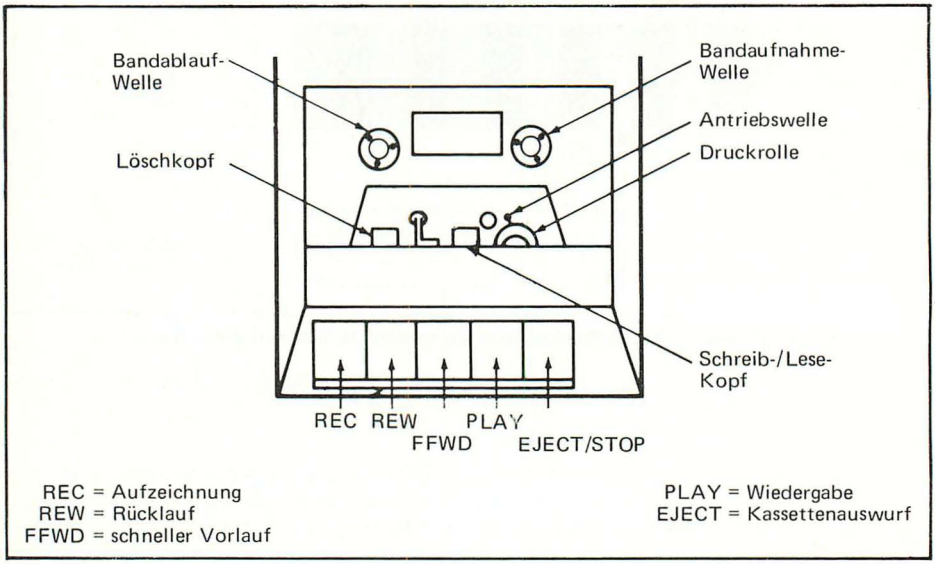
1. Schalten Sie den Computer aus.
2. Führen Sie den Anschlußstecker nach vorangegangener Sichtprüfung seitenrichtig und ohne Gewalt in den Anschluß am Computer ein, und stellen Sie sicher, daß die Verbindung mechanisch fest ist.
3. Schalten Sie den Computer wieder ein.

BETRIEBSTEST

Vor dem Arbeiten mit einem Bandgerät, sollten Sie die mechanische Bedienung des Bandgerätes über seine Tastaturen prüfen. Hier die Schritte:

1. Schalten Sie den CBM-Computer ein. Stellen Sie sicher, daß keine der Tasten am Bandgerät niedergedrückt ist und daß der Bandantrieb nicht läuft.
2. Öffnen Sie den Kassettenschacht durch Drücken der Taste STOP/EJECT. Während Sie in das Gerät schauen, sollte beim Drücken der Taste PLAY sichtbar sein, wie sich der Schreib/Lese-Kopf (Figur 1-20) nach vorne bewegt und sich gleichzeitig die Druckrolle an die Antriebswelle anlegt und im Uhrzeigersinn zu rotieren beginnt.
3. Drücken Sie die Taste STOP/EJECT. Der Schreib/Lese-Kopf sollte sich zurückbewegen und die Rotation der Antriebswelle aufhören.
4. Drücken Sie die Taste FFWD. Der Schreib/Lese-Kopf sollte unsichtbar bleiben und die Bandaufnahme-Welle auf der rechten Seite entgegen dem Uhrzeigersinn sehr schnell rotieren.
5. Drücken Sie die Taste STOP/EJECT. Die Rotation der Bandaufnahme-Welle sollte aufhören.

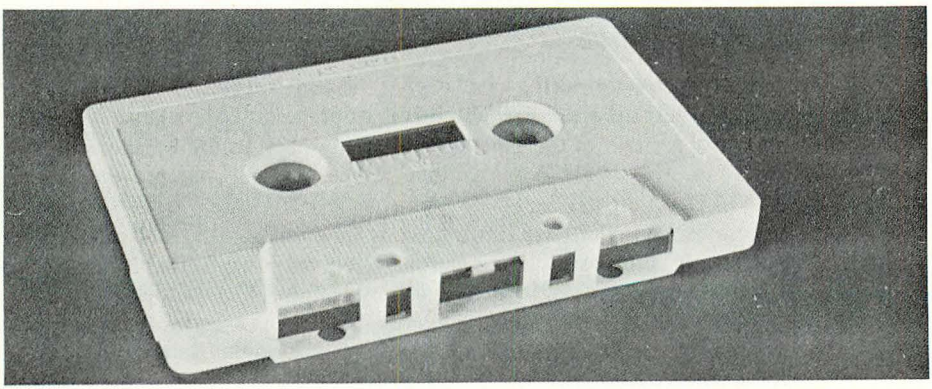
6. Drücken Sie die Taste REW. Der Schreib/Lesekopf sollte unsichtbar bleiben, während die Bandablauf-Welle auf der linken Seite im Uhrzeigersinn rotiert.
7. Drücken Sie die Taste STOP/EJECT. Die Rotation der Bandablauf-Welle sollte aufhören.
8. Drücken Sie die Taste REC. Diese Taste sollte eingerastet bleiben und sich nicht mehr bewegen. Diese Taste kann nur bedient werden, wenn zugleich die Tasten PLAY und REC niedergedrückt sind und eine Kassette eingelegt worden ist.



Figur 1-20. Einzelheiten am Bandgerät

BANDKASSETTEN

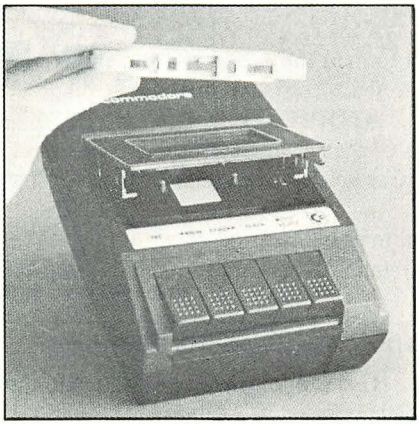
Sie werden zum Arbeiten mit Ihrem Bandgerät vielleicht Kassetten kaufen, auf die bereits Programme aufgezeichnet wurden, oder leere Kassetten, auf die Sie eigene Programme und Daten aufzeichnen (Figur 1-21).



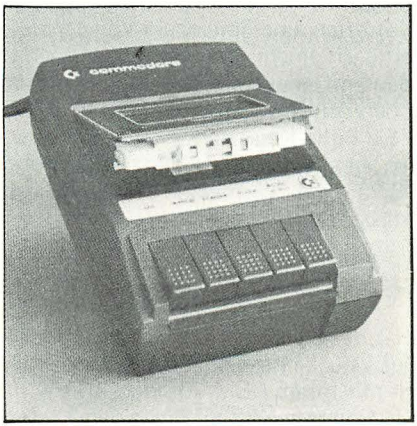
Figur 1-21. Bandkassete

Das richtige Einlegen einer Kassette in das Bandgerät zeigt die Bilderfolge in Figur 1-22.

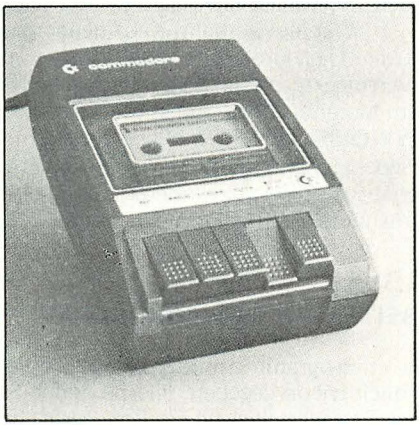
a. Leeres, geöffnetes Bandgerät



b. Richtige Stellung der Bandkassette vor dem Einlegen



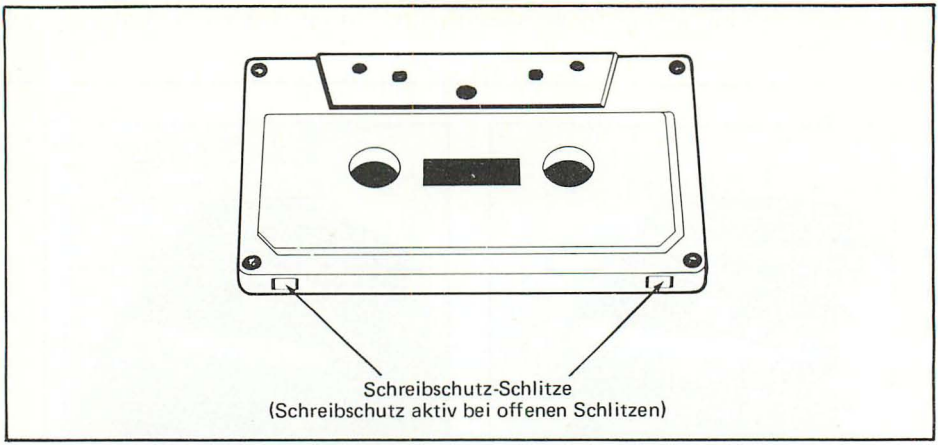
c. Eingelegte Bandkassette



d. Bandgerät geladen mit Bandkassette

Figur 1-22. Einlegen einer Bandkassette

Der Inhalt von Kassetten, z.B. Programme oder Daten, kann gegen Zerstörung durch versehentliches Überschreiben geschützt werden, wenn die Schreibschutz-Schlitze auf der Rückseite der Kassette herausgebrochen werden (Figur 1-23).



Figur 1-23. Schreibschutz an Bandkassetten

Hier einige Hinweise für den Umgang mit Kassetten. Eine neue Kassette, ob bespielt oder nicht, sollten Sie zuerst auf Ihrem Bandgerät "akklimatisieren", indem Sie das Band einem Umspul-Zyklus unterwerfen: Zuerst mit der Taste FFWD bis zum Bandende vorlaufen lassen und dann mit der Taste REW zurückspulen. Hierdurch werden Lesefehler verringert.

Wählen Sie beim Kauf neuer, leerer Kassetten eine Spieldauer von 15 bis 30 Minuten. Hierdurch wird nicht nur die Suchzeit nach einem Programm oder nach Daten verringert, sondern Sie erhalten auch ein Band mit dickerem und widerstandsfähigerem Material. Bevorzugen Sie Kassetten mit den Aufschriften HIGH QUALITY, LOW NOISE, HIGH OUTPUT und vermeiden Sie Billig-Kassetten, die häufig weniger zufriedenstellend sind. Bewahren Sie Kassetten an einem kühlen Platz abseits von magnetischen Feldern anderer elektronischer Geräte auf. Vermeiden Sie die Berührung der magnetischen Schicht des Bandes beim Einlegen.

ARBEITEN MIT DEM BANDGERÄT

Programme und Daten werden wie Musikstücke nacheinander aufgezeichnet und auch wiedergegeben. Hierbei teilen Sie sich mit Ihrem CBM-Computer die Aufgabe der Tastenbedienung am Bandgerät. Ihr CBM-Computer öffnet nicht den Kassettenschacht zum Austausch einer Kassette (Taste EJECT), sorgt nicht für das Rückspulen auf dem Bandanfang (Taste REW) oder für den schnellen Vorlauf bis zu einer gewünschten Aufzeichnung (FFWD). Vor Beginn einer Aufzeichnung gibt Ihnen jedoch Ihr CBM-Computer auf dem Bildschirm die Nachricht "PRESS PLAY AND RECORD" aus, womit Sie aufgefordert werden, die Tasten REC und PLAY zu drükken.

Die Taste REW betätigen Sie, um das Magnetband mit hoher Geschwindigkeit zurückzuspulen, z.B. am Ende einer Kassettenbenutzung oder vor Beginn des Suchvorgangs Ihres Computers nach einer Aufzeichnung von Programmen oder Daten auf dem Band.

Die Taste FFWD dient zum schnellen Bandvorlauf. Von dieser Möglichkeit machen Sie Gebrauch, wenn Sie den Suchvorgang des Computers nach Aufzeichnungen auf dem Band verkürzen wollen, indem Sie das Band bis zu der gewünschten Aufzeichnung vorlaufen lassen.

Die Taste PLAY löst eine geringere Laufgeschwindigkeit des Bandes aus, bei der der Computer nach Programmen oder Daten auf dem Band suchen und sie in seinen Speicher laden kann.

Die Taste REC mit nachfolgend gedrückter Taste PLAY ermöglicht dem Computer, ein Programm oder Daten aus seinem Speicher auf das Band zu schreiben. Der Zeitpunkt dieser Tastenbedienung wird vom Computer bestimmt und durch eine Aufforderung auf dem Bildschirm angezeigt.

CBM FLOPPY DISK-SPEICHER

Das physikalische Prinzip der Aufzeichnung von Programmen und Daten ist bei Floppy Disk-Speichern das gleiche wie bei Bandgeräten; statt eines Bandes werden jedoch dünne, flexible Scheiben mit magnetisierbarer Beschichtung von der Größe einer Single-Schallplatte verwendet, für die Namen wie "Diskette" oder "Floppy" gebräuchlich sind.

In Figur 1-24 finden Sie Abbildungen der Floppy Disk-Speicher CBM 3040, CBM 4040, CBM 8050. Diese Geräte sind mit Doppellaufwerken ausgerüstet, so daß auf jedem Gerät die Speicherkapazität von zwei Disketten zur Verfügung steht. Die Technischen Daten dieser Geräte sind in Tabellen 1-2 und 1-3 zusammengestellt.

CBM 3040 176 640 Bytes/Diskette
BASIC <3.0, DOS 1.0

CBM 4040 174 848 Bytes/Diskette
BASIC 4.0, DOS 2.0

CBM 8050 533 248 Bytes/Diskette
BASIC 4.0, DOS 2.5

Figur 1-24. CBM Floppy Disk-Speicher (Doppellaufwerke)

Tabelle 1-2. Technische Daten des CBM 8050

Stahlgehäuse
Höhe: 18 cm
Breite: 38 cm
Tiefe: 40 cm

Schaltkreise
Steuereinheit:
6502 Mikroprozessor
6530 E/A RAM, ROM
6522 E/A, Taktgeber
Interface:
6502 Mikroprozessor
6532 (2) E/A RAM, Taktgeber
6564 (2) ROM
Gemeinsame Speicher:
6114 (8) 4 x 1K RAM

Betriebssystem
DOS 2.5

Elektrische Daten

Spannung	: 220 V
Frequenz	: 50 Hz
Leistungsaufnahme	: 50 W

Laufwerke

Shugart	: SA390 (2)
Disketten	: 5 1/4'' soft-sektoriert einseitig

Speicherkapazität je Diskette

Gesamtkapazität	: 533 248 Bytes
Sequent. Dateien	: 521 208 Bytes
Relative Dateien	: 464 312 bis 517 398 Bytes*)
Sektoren je Spur	: 23 bis 29
Bytes je Sektor	: 256
Spuren	: 77
Sektoren	: 2083

*) abhängig von dern Dateilängen (s. Kapitel 6)

Tabelle 1-3. Technische Daten des CBM 3040, 4040

Stahlgehäuse
Höhe: 18 cm
Breite: 38 cm
Tiefe: 40 cm

Schaltkreise
Steuereinheit:
6504 Mikroprozessor
6530 E/A RAM, ROM
6522 E/A Taktgeber
Interface:
6502 Mikroprozessor
6532 (2) E/A RAM, Taktgeber
6332 (2) ROM
Gemeinsame Speicher:
6114 (8) 4 x 1K RAM

Betriebssystem
CBM 3040: DOS 1
CBM 4040: DOS 2

Elektrische Daten

Spannung	: 220 V
Frequenz	: 50 Hz
Leistungsaufnahme	: 50 W

Laufwerke

Shugart	: SA390 (2)
Disketten	: 5 1/4 soft-sektoriert einseitig

Speicherkapazität je Diskette

CBM 3040

Gesamtkapazität	: 176 640 Bytes
Sequent. Dateien	: 170 180 Bytes
Wahlfreie Dateien	: 170 850 Bytes

CB 4040

Gesamtkapazität	: 174 848 Bytes
Sequent. Dateien	: 168 656 Bytes
Relative Dateien	: 167 132 bis 65535 Bytes*

CBM 3040, 4040

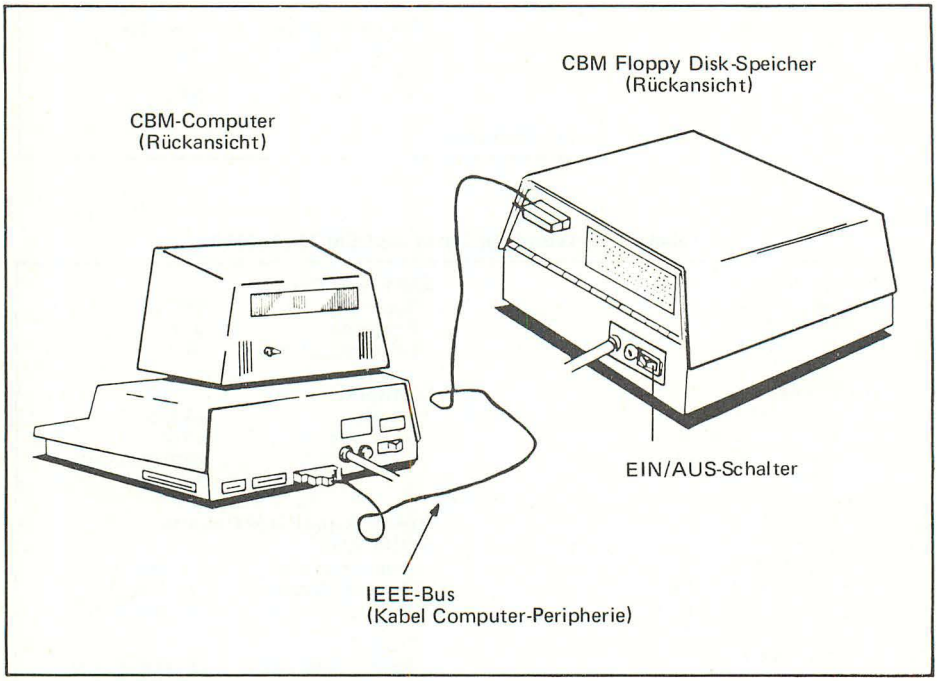
Sektoren je Spur	: 17 bis 21
Bytes je Sektor	: 256
Spuren	: 35
Sektoren	: 690 (CBM 3040) 683 (CBM 4040)

*) abhängig von den Dateilängen (s. Kapitel 6)

ANSCHLUSS

Gehen Sie zum Anschluß eines Floppy Disk-Speichers an Ihren CBM-Computer in folgenden Schritten vor:

1. Ziehen Sie den Netzstecker zu Ihrem Computer aus der Steckdose.
2. Stellen Sie mit einem Kabel Computer-Peripherie die Verbindung zwischen dem IEEE-Anschluß J1 und dem Floppy Disk-Speicher her (Figur 1-25). Beachten Sie, daß die Steckerverbindung mit dem Floppy Disk-Speicher durch Anziehen von zwei gerändeten Schrauben hergestellt wird.
3. Schließen Sie die beiden Geräte über ihre Netzstecker an eine geerdete Steckdose an.
   Nach Herstellung der Kabelverbindung folgt der Einschalttest.



Figur 1-25. Anschluß eines Floppy Disk-Speichers

EINSCHALTTEST

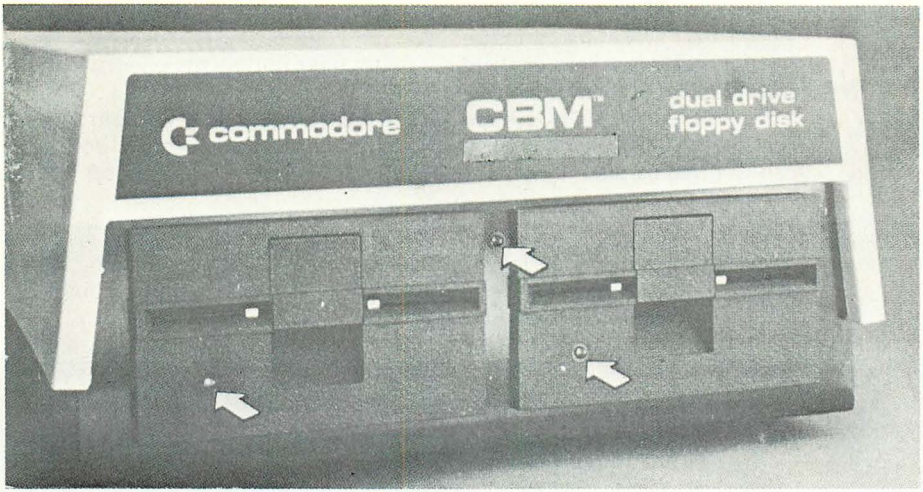
1. Schalten Sie zuerst den CBM-Computer ein. Prüfen Sie, ob Tastatur und Bildschirm Ihres Computers richtig arbeiten.
2. Für Modelle CBM 3040 und CBM 4040: Öffnen Sie die Ladeklappen an beiden Laufwerken (Figur 1-26). Überspringen Sie diesen Schritt für das Modell CBM 8050.
3. Stellen Sie sicher, daß beide Laufwerke leer sind, d.h. nicht mit einer Diskette geladen.
4. Schalten Sie den Floppy Disk-Speicher über seinen Wippschalter auf der Rückseite des Gerätes ein.

5. CBM 8050: Auf der Frontseite des Gerätes sollten alle drei grünen Anzeigelichter zweimal aufleuchten. Während das mittlere Anzeigelicht weiterhin leuchtet, sollten die beiden Lichter links und rechts oberhalb der Laufwerke ausgehen.
   CBM 3040 und CBM 4040: Auf der Frontseite des Gerätes sollten alle drei roten Anzeigelampen kurz aufleuchten und dann ausgehen. Bei einigen Geräten ist diese Einschaltphase mit einem surrenden Geräusch der Laufwerke verbunden.
6. Wenn Anzeigelampen nicht erlöschen, schalten Sie den Floppy Disk-Speicher wieder aus. Warten Sie 5 Minuten und wiederholen Sie die Einschaltung. Wenn die Lampen weiterhin leuchten, nehmen Sie Kontakt mit Ihrem Commodore-Händler auf.

Figur 1-26. Öffnen der Ladeklappe (CBM 3040 und 4040)

ANZEIGELAMPEN

CBM Floppy Disk-Speicher sind auf ihrer Frontseite mit 3 Anzeigelampen in Form von Leuchtdioden ausgerüstet (Figur 1-27). Laufwerk 0 (drive 0) und Laufwerk 1 (drive 1) haben ihre eigenen Leuchtdioden, die aufleuchten, sobald das betreffende Laufwerk in Betrieb ist. Die Leuchtdiode in der Mitte der Frontseite dient zur Fehleranzeige.



Figur 1-27. Leuchtdioden auf der Frontseite von Floppy Disk-Speichern

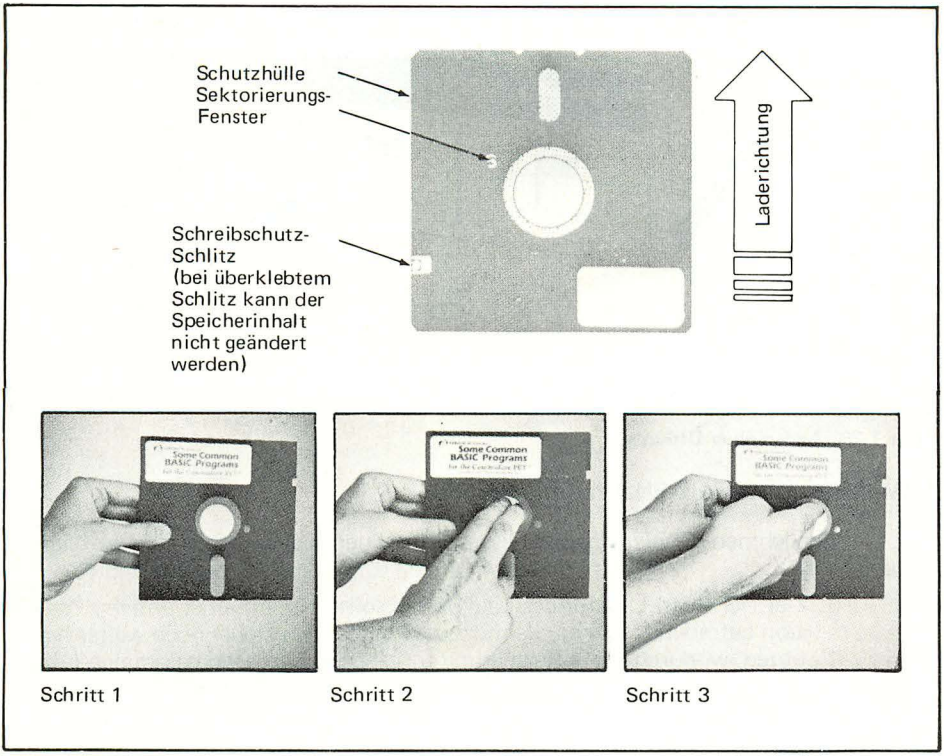
Die Floppy Disk-Speicher CBM 8050 sind mit grünen Leuchtdioden ausgestattet. Während des Betriebs eines Laufwerks leuchtet die zugehörige Leuchtdiode auf. Die mittlere Leuchtdiode dient als Fehleranzeige: Bei Inbetriebnahme des CBM 8050 leuchtet sie grün auf und wechselt ihre Farbe zur Fehleranzeige in rot. Das rote Licht bleibt bestehen, bis der Fehler behoben ist.

Die Floppy Disk-Speicher CBM 3040 und CBM 4040 sind mit roten Leuchtdioden ausgestattet. Der Betrieb eines Laufwerks wird durch Aufleuchten seiner zugeordneten Leuchtdiode angezeigt. Die Leuchtdiode in der Mitte dient alleine zur Fehleranzeige; sie leuchtet im Fehlerfall auf und bleibt eingeschaltet, bis der Fehler korrigiert ist.

UMGANG MIT DEN DISKETTEN

Jede Diskette ist in eine rechteckige Schutzhülle der Abmessung 13 x 13 cm eingeschweißt und wird samt der Schutzhülle in das Laufwerk eines Floppy Disk-Speichers eingelegt (Figur 1-28). Für Ihren CBM Floppy Disk-Speicher kaufen Sie soft-sektorierte 5 1/4 Zoll Disketten. Die Art der Sektorierung können Sie in folgenden Schritten prüfen (Figur 1-28):

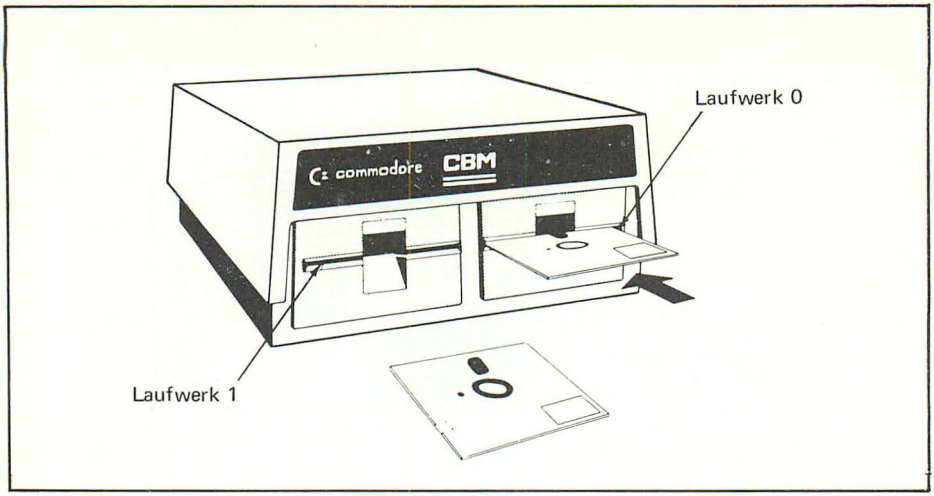
1. Drehen Sie mit zwei Fingern die Diskette innerhalb Ihrer Schutzhülle, wobei Sie das Sektorierungs-Fenster beobachten.
2. Dis Diskette ist soft-sektoriert, wenn bei der Rotation nur ein kleines Loch in der Diskette über das Sektorierungsfenster sichtbar wird. Erschei nen mehrere Löcher unter dem Sektorierungsfenster, dann ist Ihre Diskette nicht soft-sektoriert und nicht für CBM Floppy Disk-Speicher geeignet.



Figur 1-28. Diskette und Prüfung der Art ihrer Sektorierung (Schritte 1 - 3)

CBM 8050: LADEN EINER DISKETTE

1. Greifen Sie die Diskette an ihrer Schutzhülle und vermeiden Sie die Berührung an den ungeschützten Teilen ihrer Oberfläche. Die Etikett-Seite der Diskette sollte oben und der Schreibschutz-Schlitz auf der linken Seite liegen.
2. Führen Sie die Diskette vorsichtig in einen der Schlitze ein (Figur 1-29), bis Sie ein lautes Klicken hören. Versuchen Sie nicht, die Diskette weiter hineinzuschieben und vermeiden Sie jede Kraftanwendung: Sie könnten sowohl die Diskette als auch das Laufwerk beschädigen.
3. Drücken Sie mit zwei Fingern den Hebel im Ladetor des Laufwerks nieder, bis er in der unteren Stellung bleibt.

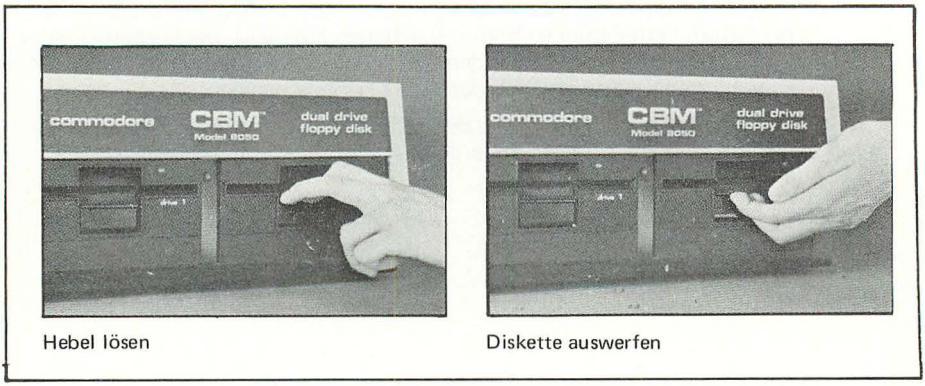


Figur 1-29. Laden einer Diskette

CBM 8050: ENTNEHMEN EINER DISKETTE

Entnehmen Sie niemals eine Diskette aus einem Laufwerk, dessen Anzeigelampe brennt.

1. Der Hebel am Ladetor des Laufwerkes sollte sich in seiner unteren Position befinden. Drücken Sie mit zwei Fingern den Hebel noch weiter nach unten, wodurch seine Arretierung aufgehoben wird und zusammen mit seiner Aufwärtsbewegung die Diskette im Schlitz des Laufwerks erscheint (Figur 1-30).
2. Zum Auswurf der Diskette legen Sie Ihren Zeigefinger unter den Hebel und drücken ihn vorsichtig nach oben und vorne. Hierdurch wird die Diskette aus dem Schlitz des Laufwerks ausgeworfen (Figur 1-30).
3. Greifen Sie die Diskette zwischen Daumen und Zeifefinger und ziehen Sie sie vorsichtig aus dem Laufwerk. Vermeiden Sie jede Verbiegung oder gewaltsame Bewegung.
4. Stecken Sie die Diskette in den Schutzumschlag, indem sie sich beim Kauf befand.



Hebel lösen

Diskette auswerfen

Figur 1-30. Entnahme einer Diskette (CBM 8050)

CBM 3040, 4040: LADEN EINER DISKETTE

Schalten Sie niemals einen Floppy Disk-Speicher ein, dessen Laufwerke noch mit Disketten geladen sind.

1. Vergewissern Sie sich, daß der Floppy Disk-Speicher nicht eingeschaltet ist. Öffnen Sie die Ladeklappen an den Laufwerken, um zu prüfen, daß kein Laufwerk mit einer Diskette geladen ist (Figur 1-26). Danach erst schalten Sie Ihren Floppy Disk-Speicher ein.
2. Führen Sie die Diskette, wie in Figur 1-29 gezeigt, in den Schlitz des Laufwerks ein. Der Schreibschutz-Schlitz an der Diskette befindet sich hierbei auf der linken Seite. Verwenden Sie keine besondere Kraft für diesen Vorgang, Sie könnten die Diskette oder das Laufwerk beschädigen.
3. Schließen Sie das Laufwerk durch Niederdrücken der Ladeklappe, bis Sie ein leichtes Klick hören. Die Diskette ist jetzt für den Betrieb richtig positioniert.

CBM 3040, 4040: ENTNAHME EINER DISKETTE

Entnehmen Sie niemals einem Laufwerk mit brennender Anzeigelampe die Diskette.

1. Öffnen Sie die Ladeklappe am Laufwerk mit dem Zeigefinger (Figur 1-26).
2. Greifen Sie die Diskette mit Daumen und Zeigefinger und entnehmen Sie sie vorsichtig dem Laufwerk, ohne sie zu biegen oder Kraft anzuwenden.
3. Schließen Sie die Ladeklappe zum Laufwerk nach Entnahme der Diskette.
4. Stecken Sie die Diskette in den Schutzumschlag zurück, in dem Sie sie gekauft haben.

BEHANDLUNG DER DISKETTEN

Sie müssen Disketten mit Vorsicht behandeln. Bei Beschädigung einer Diskette ist es nicht mehr möglich, die darauf gespeicherten Daten zu retten. Beachten Sie folgende Hinweise zur Behandlung Ihrer Disketten.

1. Legen Sie eine Diskette sofort nach Entnahme aus einem Laufwerk in eine der Schutzhüllen aus Papier, mit denen die Disketten geliefert wurden. Hierdurch wird die Oberfläche der Diskette vollständig gegen Staub geschützt.
2. Entfernen Sie die Diskette niemals aus ihrer Plastik-Schutzhülle.
3. Benutzen Sie zur Beschriftung einer Diskette die mitgelieferten Etiketten. Schreiben Sie niemals auf das Etikett mit einem Kugelschreiber oder Bleistift; verwenden Sie nur Filzstifte.
4. Berühren Sie nicht die Oberfläche der Diskette und versuchen Sie auch nicht, diese Oberfläche zu reinigen.
5. Rauchen Sie nicht bei Benutzung von Disketten. Tabakasche und Rauchniederschläge auf der Oberfläche der Diskette können sie zerstören.
6. Bewahren Sie Disketten abseits von anderen elektrischen Geräten auf, de-

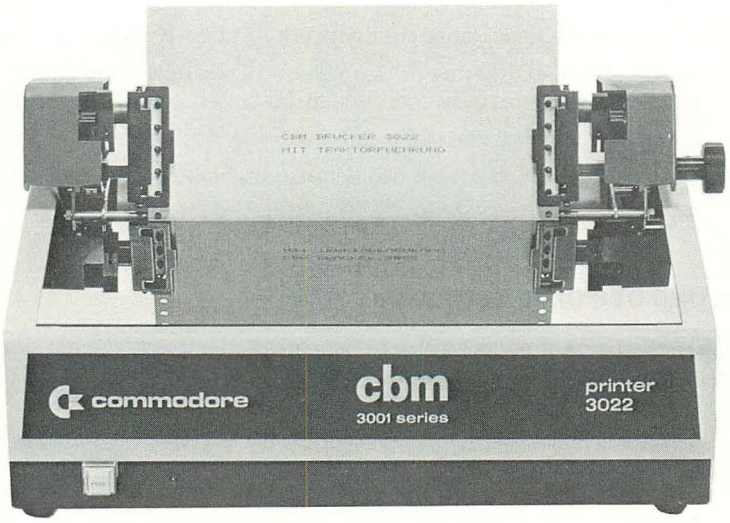
ren Magnetfelder die gespeicherten Daten auf der Diskette zerstören können.

7. Vermeiden Sie Hitze und Sonnenbestrahlung.

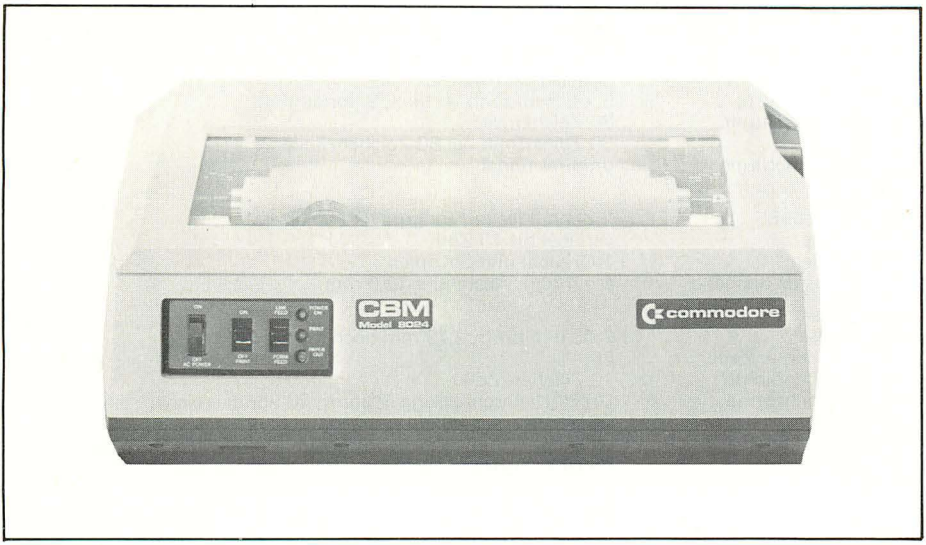
CBM-TEXTDRUCKER

In diesem Buch wird das Arbeiten mit folgenden CBM-Textdruckern beschrieben:

1. CBM 3022 (Figur 1-31, Tab. 1-4)
2. CBM 4022 (Tab. 1-5)
3. CBM 8024 (Figur 1-33, Tab. 1-6)



Figur 1-31. Drucker CBM 3022



Figur 1-32. Drucker CBM 8024

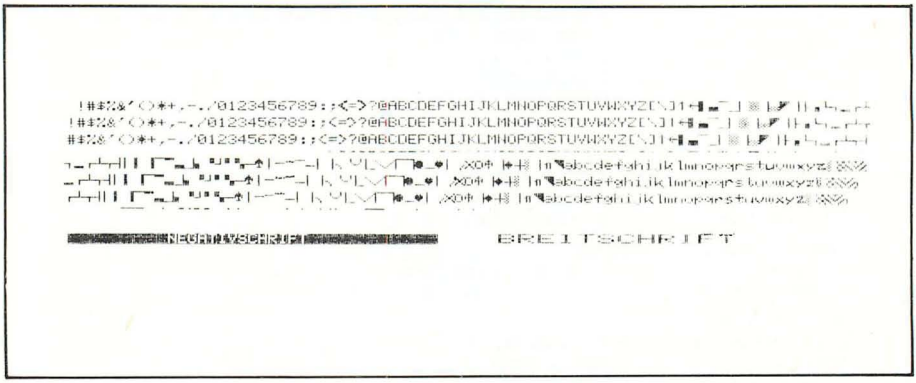
Tabelle 1-4. Technische Daten des CBM 3022

Druck	
Verfahren	: Serieller Matrixdrucker mit Nadeldruckkopf
Druckmatrix	: 6 x 7 Punkte (6 vertikal, 7 horizontal)
Druckleistung	: 150 Zeichen/sec 70 Zeilen/min bei 80 Zeichen/Zeile
Druckrichtung	: unidirektional
Papier	
Vorschub	: 2 Zahnräder mit Eingriff im Papierrand
Breite	: variabel bis 10 Zoll
Kopien	: 1 Original und 2 Kopien
Transportlöcher	: ϕ = 4 mm, Abstand = 12,6 mm
Zeichen	
Größe	: 2,54 mm breit, 2,79 mm hoch
Zeichen je Zeile	: 80
Zeichendichte	: 10 Zeichen/Zoll
Zeichenformat	: Groß-/Kleinschreibung, Ziffern, 26 Sonderzeichen, alle CBM-Graphiksymbole
Sonderzeichen	: beliebig programmierbare Sonderzeichen Negativschrift Breitschrift Sperrschrift
Druckbild	
Zeilenabstand	: programmierbar
Zeilen je Seite	: programmierbar
Druckformate	: programmierbar
Typische Gerätekombinationen	
Computer	: CBM 3001/4001-Serie
Floppy Disk-Speicher	: CBM 3040 oder CBM 4040
Elektrische Daten	
Computer-Anschluß	: IEEE-488-Bus
Netzanschluß	: 220 V, 50 Hz
Leistungsaufnahme	: 200 W max.

Tabelle 1-5. Technische Daten des CBM 4022

Druck	
Verfahren	: Serieller Matrixdruck
Druckmatrix	: 5 x 8 Punkte (5 vertikal, 8 horizontal)
Druckleistung	: 80 Zeichen/sec 30 Zeilen/min bei 80 Zeichen/Zeile
Druckrichtung	: unidirektional
Papier	
Vorschub	: 2 Zahnräder mit Eingriff im Papierrand
Breite	: variabel bis 10 Zoll
Kopien	: 1 Original und 2 Kopien
Transportlöcher	: ϕ = 4 mm, Abstand = 12,6 mm
Zeichen	
Größe	: 2,03 mm breit, 2,79 mm hoch
Zeichen je Zeile	: 80
Zeichendichte	: 10 Zeichen/Zoll
Zeichenformat	: Groß-/Kleinschreibung, Ziffern, 26 Sonderzeichen, alle CBM-Graphiksymbole
Sonderzeichen	: selbstentworfene Zeichen programmierbar Negativschrift Breitschrift Sperrschrift
Druckbild	
Zeilenabstand	: programmierbar
Zeilen je Seite	: programmierbar
Druckformate	: programmierbar
Typische Gerätekombinationen	
Computer	: CBM 4001-Serie oder CBM 8032
Floppy Disk-Speicher	:CBM 4040
Elektrische Daten	
Computer-Anschluß	: IEEE-488-Bus
Netzanschluß	: 220 V, 50 Hz
Leistungsaufnahme	: 200 W max.

*) für Schriftbilder siehe folgenden Abschnitt



Figur 1-33. Schriftbild des Druckers CBM 4022

Tabelle 1-6. Technische Daten des CBM 8024

Druck	: Serieller Matrixdruck
Verfahren	: 7 x 7 Punkte (9 x 9 wahlweise)
Druckleistung	: 160 Zeichen/sec
Druckrichtung	: bidirektional
Papiervorschub	
Prinzip	: 2 Traktorantriebe
Formblattvorschub	: 22 cm/sec
Zeilenvorschub	: 70 cm/sec
Papier	
Breite	: 10 bis 39 cm, verstellbar
Kopien	: 1 Original, 5 Kopien
Transportlöcher	: ϕ = 4 mm, Abstand = 12,6 mm
Zeichen	
Größe	: 2,03 mm breit, 2,69 mm hoch
Zeichen je Zeile	: bis 132 (bei 10 Zeichen/Zoll)
Zeichendichte	: 10 Zeichen/Zoll
Zeichenvorrat*	: Groß-/Kleinschreibung, Ziffern, Sonderzeichen
Sonderzeichen	: . bis zu 96 Benutzer-Zeichen . Negativschrift . Breitschrift . Sperrschrift
Druckbild	
Zeilenabstand	: 6 oder 8 Zeilen/Zoll
Zeilen je Seite	: programmierbar
Druckformate	: programmierbar
Typische Gerätekombinationen	
Computer	: CBM 4032 oder CBM 8032
Floppy Disk-Speicher	: CBM 4040, CBM 8050
Elektrische Daten	
Computer-Anschluß	: IEEE-488-Bus
Netzanschluß	: 220 V, 50 Hz
Leistungsaufnahme	: 200 W max.

*) für Schriftbilder siehe folgenden Abschnitt

Die Technischen Daten dieser Drucker finden Sie in den Tabellen 1-4, 1-5 und 1-6. Gemeinsames Kennzeichen der Drucker sind der Papiervorschub durch Zahnräder, die in den Lochrand des Druckerpapiers eingreifen, und die Verwendung von Nadeldruckköpfen, die ähnlich wie bei den Bildschirmen Zeichen in einem Punkt-Raster darstellen. Für das Schriftbild dieser Drucker finden Sie Beispiele in Figur 1-33 und 1-34.

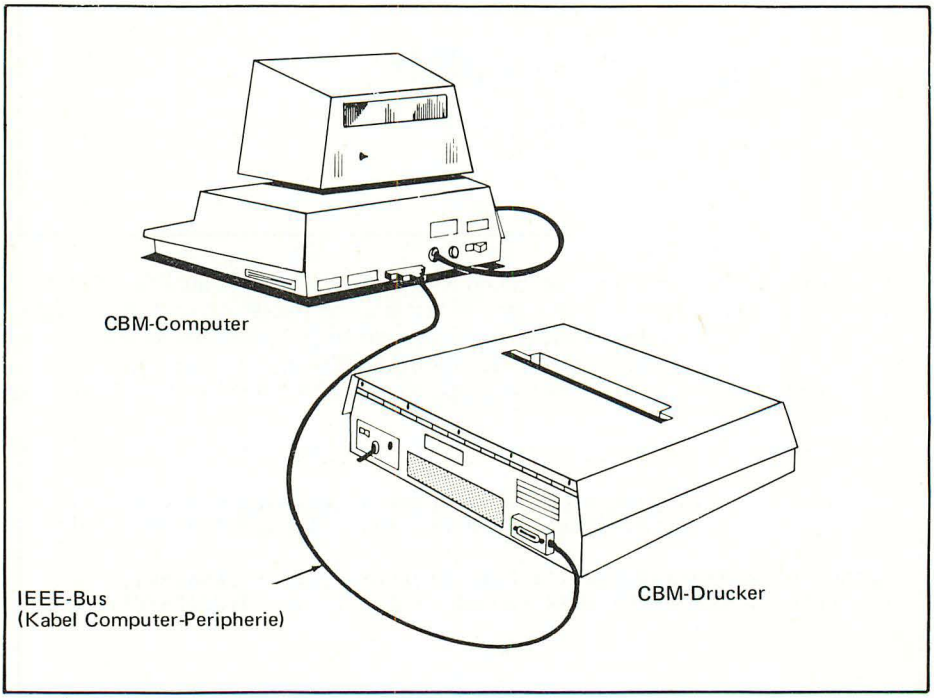
```
 !"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz[\]↑←
 !"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz[\]↑←

`ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ▒!"#$%&'()*+,-./0123456789:;<=
 ←`ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ▒!"#$%&'()*+,-./0123456789:;
```

Figur 1-34. Schriftbild des Druckers CBM 8024 (Auszug)

ANSCHLUSS

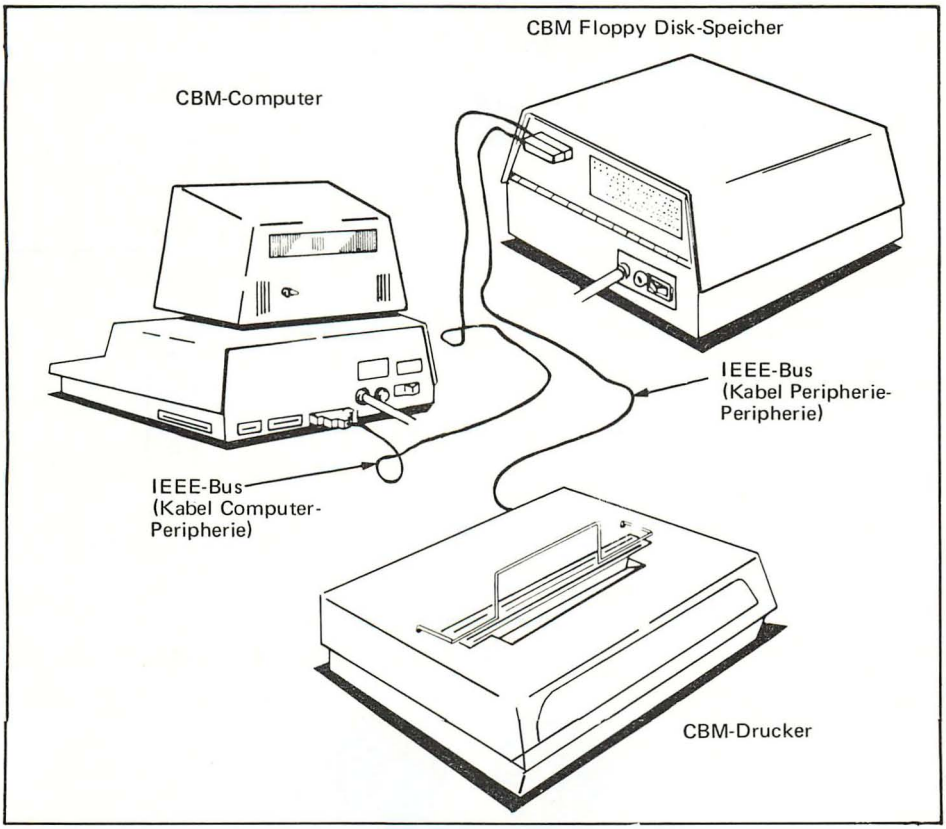
Wenn der Drucker als einziges .Gerät an Ihren Computer angeschlossen werden soll, gehen Sie in folgenden Schritten vor (Figur 1-35):

1. Ziehen Sie den Netzstecker Ihres Computers aus der Steckdose.
2. Stellen Sie mit einem Kabel "Computer-Peripherie" die IEEE-488-Bus-Verbindung zwischen Computer und Drucker her. Befestigen Sie den Kabelstecker am Drucker durch Verschraubung. Achten Sie auf das seitenrichtige Einführen des anderen Kabelsteckers in den Anschluß J1 für den IEEE-488-Bus auf der Rückseite Ihres Computers.
3. Nehmen Sie jetzt Ihren Computer wieder in Betrieb. Prüfen Sie kurz, ob Tastenfeld und Bildschirm richtig arbeiten.
4. Verbinden Sie den Drucker mit der Netzsteckdose. Schalten Sie den Drucker über seinen Wippschalter auf der rechten Rückseite des Druckers ein. Als erste Reaktion des Druckers sollte sich der Druckkopf ganz nach rechts und dann zurück zu seiner Ausgangsstellung bewegen.
5. Wiederholen Sie die einzelnen Schritte des Druckeranschlusses, wenn sich der Druckkopf beim Einschalten nicht bewegen sollte. Überprüfen Sie insbesondere noch einmal die Steckerverbindungen mit dem Computer und mit dem Drucker.



Figur 1-35. Anschluß eines Druckers

Wenn der Drucker zusammen mit einem Floppy Disk-Speicher an Ihrem Computer betrieben werden soll, müssen Sie die in Figur 1-36 gezeigte Reihenfolge des Geräteanschlusses einhalten. Der Floppy Disk-Speicher wird über ein Computer-Peripherie-Kabel mit dem Anschluß J1 für den IEEE-488-Bus verbunden. Diese Busverbindung wird über ein Kabel Peripherie-Peripherie bis zum Drucker fortgesetzt: Der eine Kabelstecker wird für eine Huckepack-Verbindung auf den Stecker am Floppy Disk-Speicher gesteckt und verschraubt, der andere Kabelstecker durch Verschraubung mit dem Drucker verbunden.

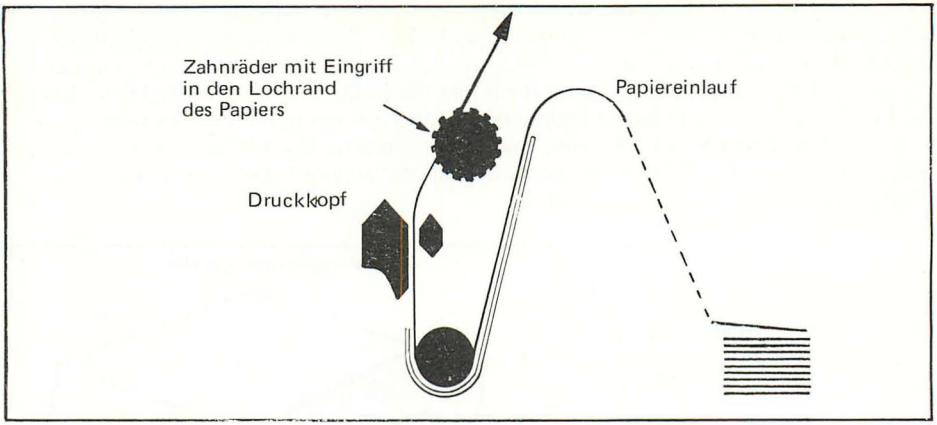


Figur 1-36. Anschluß eines Druckers bei gleichzeitig angeschlossenem Floppy Disk-Speicher

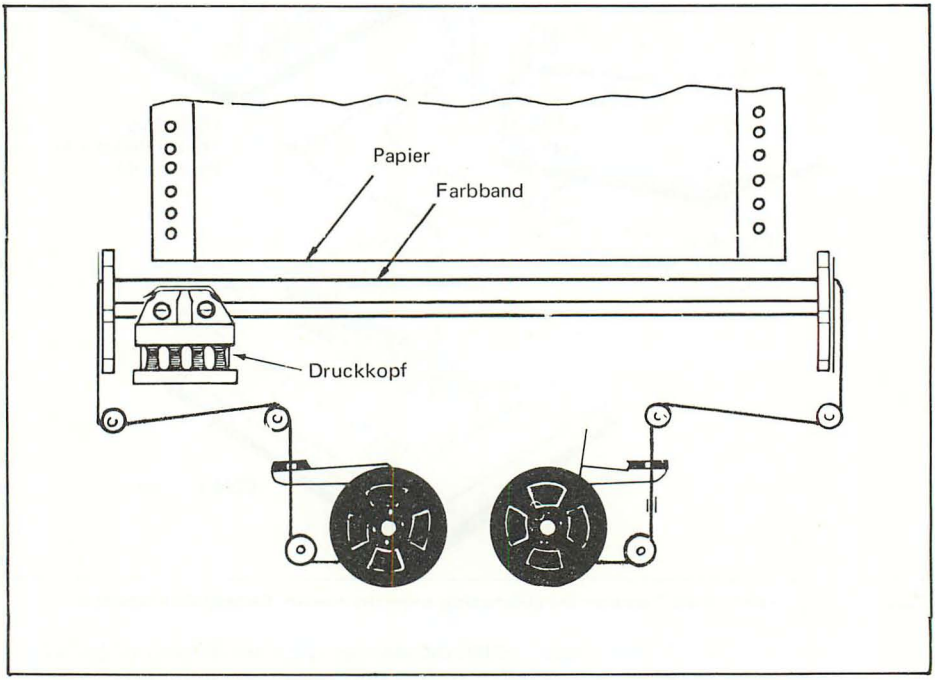
Auch bei dieser Anschlußart gelten die eben genannten Schritte: Während der Herstellung der Kabelverbindung bleiben alle Geräte vom Netz getrennt. Danach wird als erstes Gerät Ihr Computer eingeschaltet, dann der Drucker.

EINLEGEN VON PAPIER UND FARBBAND

Der Papierverlauf im Drucker und die Ausführung des Farbbandes weisen zwischen den einzelnen CBM-Druckern leichte Unterschiede auf. Die folgenden Abbildungen geben Ihnen eine Vorstellung vom Verlauf des Papiers und des Farbbandes im Drucker (Figur 1-37 und Figur 1-38).



Figur 1-37. Typischer Papierverlauf im Drucker



Figur 1-38. Typischer Verlauf des Farbbands im Drucker

FUNKTIONSTEST DES DRUCKKOPFES

Alle CBM-Drucker können einen Selbsttest ausführen, in dessen Verlauf sämtliche, vom Drucker darstellbaren Zeichen ausgedruckt werden.

Starten Sie diesen Selbsttest erst, nachdem Sie Farbband und Papier eingelegt haben; sie riskieren sonst die Beschädigung des Druckkopfes.

Gehen Sie zum Auslösen des Selbsttestes wie folgt vor:

1. Nehmen Sie zuerst Ihren CBM-Computer in Betrieb.
2. Halten Sie die Taste für Papiervorlauf gedrückt, während Sie Ihren Drukker einschalten. Der Drucker sollte hierauf mit dem Ausdrucken seiner verfügbaren Zeichen beginnen (Figur 1-39).
3. Schalten Sie zum Unterbrechen des Selbsttests den Drucker wieder aus.

Figur 1-39. Ausdrucke beim Selbsttest eines Druckers (Auszug)

KAPITEL 2

Sofort arbeiten mit CBM Geräten

Dieses Kapitel erklärt Ihnen, wie Sie vom Tastenfeld aus sofort mit Ihrem CBM-Computer, CBM-Floppy-Disk-Speicher, CBM-Bandkassetten-Gerät und CBM-Drucker arbeiten können.

Ihre Aufträge teilen Sie einem CBM-Computer durch "Anweisungen" mit. Anweisungen werden in einer einfachen Sprache, genannt CBM BASIC, formuliert und dann vom Computer ausgeführt.

Eine Anweisung in CBM BASIC kann eine Länge von 80 Zeichen haben. 80 Zeichen entsprechen zwei Bildschirmzeilen auf 40spaltigen Bildschirmen und einer Bildschirmzeile auf 80spaltigen Bildschirmen. Beenden Sie die Eingabe einer Anweisung durch Drücken der Taste RETURN. Bei 40spaltigen Bildschirmen schreiben Sie einfach über das Ende der ersten Bildschirmzeile hinaus: Der Kursor springt automatisch nach dem 40sten Zeichen auf die nächste Bildschirmzeile. Erst dort beenden Sie die Eingabe Ihrer Anweisung mit RETURN. Wird Ihre Anweisung länger als 80 Zeichen, dann meldet sich der CBM-Computer mit der Nachricht ?SYNTAX ERROR, d.h. einer Fehlermeldung.

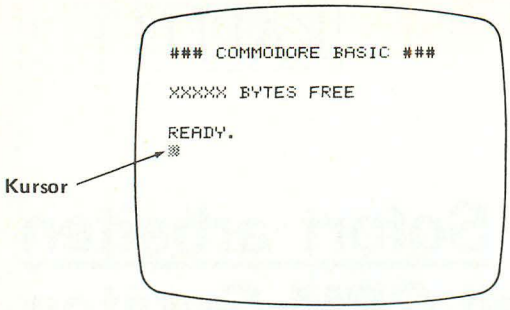
Die Eingabe von RETURN am Ende jeder Anweisung versteht der CBM-Computer als Zeichen, daß die Anweisung vollständig ist und ausgeführt werden soll.

Zur Eingabe von Anweisungen gibt es zwei Möglichkeiten: den direkten Dialog oder die Programmform. In direktem Dialog werden Anweisungen unmittelbar nach Eingabe von RETURN ausgeführt: daher der Name "Direkter Dialog". Derartige Anweisungen sind in der Regel kurz und werden nicht im Computer gespeichert. In Form eines Programms eingegebene Anweisungen werden zunächst im Computer gespeichert und erst ausgeführt, wenn Sie den Computer ausdrücklich hierzu auffordern.

Wir werden zuerst das Arbeiten in direktem Dialog und danach die Eingabe von Programmen beschreiben.

DIREKTER DIALOG

Unmittelbar nach Inbetriebnahme ist ein CBM-Computer bereit für den direkten Dialog. Dieser Zustand wird durch einen blinkenden Kursor angezeigt:



Ein CBM-Computer verläßt nur zur Ausführung eines Programms den Zustand des direkten Dialogs. Nach jeder Programmausführung kehrt er zu diesem Zustand zurück.

ZULÄSSIGE EINGABEN

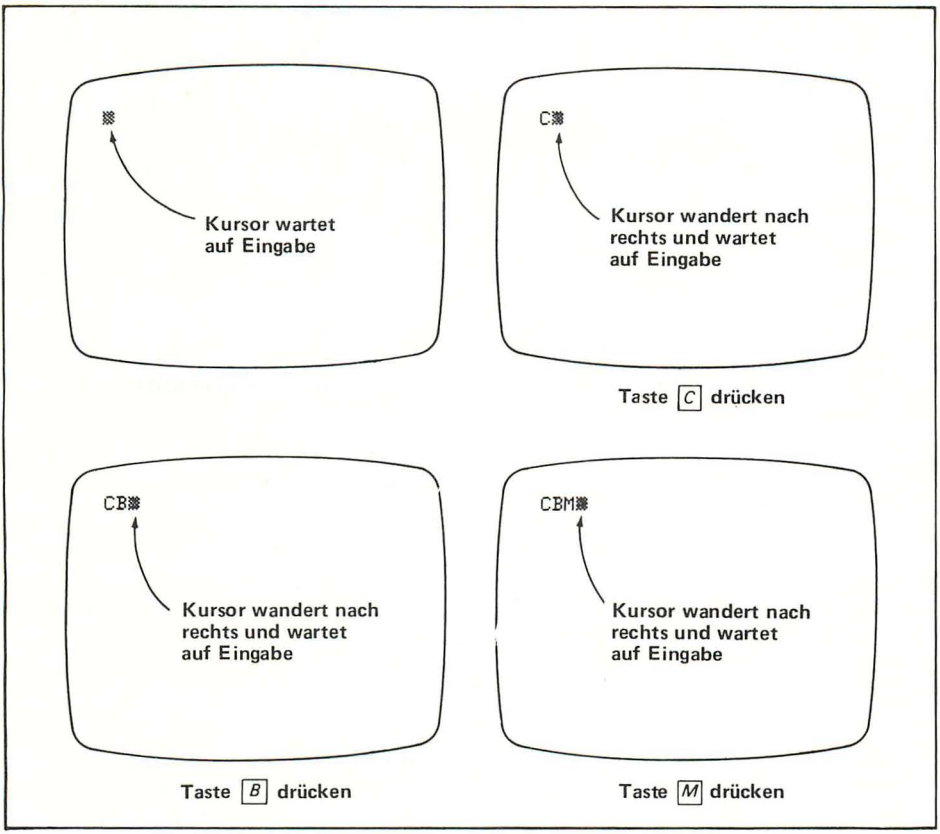
Der blinkende Kursor zeigt auf dem Bildschirm die Stelle an, an der die nächste Tastenfeldeingabe erscheinen wird. Nach jeder Eingabe eines Zeichens wandert der Kursor um eine Stelle nach rechts, wo er auf die nächste Eingabe wartet, wie Figur 2-1 zeigt. Bei den Tasten CURSOR↕ und CURSOR↔ und HOME bewegt sich der Kursor, ohne daß ein Zeichen geschrieben wird. Bei Eingabe der Taste RETURN springt der Kursor auf den Anfang der nächsten Bildschirmzeile.

Sie dürfen beliebige Zeichenfolgen über das Tastenfeld eingeben. Beenden Sie jedoch die Eingabe mit RETURN, dann hält der CBM-Computer die eingegebene Zeichenfolge für eine Anweisung, die er nach den Regeln der Sprache CBM BASIC zu interpretieren versucht. Verstößt eine eingegebene Zeichenfolge gegen die Sprache CBM BASIC, dann meldet sich der Computer mit der Fehlermeldung ?SYNTAX ERROR.

Figur 2-2 zeigt gültige Anweisungen für die Eingabe in direktem Dialog. Nach Drücken der Taste RETURN werden diese Anweisungen sofort ausgeführt und Ergebnisse, falls vorhanden, auf der nächsten Bildschirmzeile dargestellt.

Figur 2-3 zeigt unzulässige Eingaben in direktem Dialog, die mit einer Fehlermeldung ?SYNTAX ERROR quittiert werden.

Wenn Sie nur beliebigen Text und keine Anweisungen eingeben wollen, ignorieren Sie einfach die Fehlermeldungen ?SYNTAX ERROR des Computers. Eine Ausnahme bilden graphische Symbole: Der Computer wehrt sich nicht mit Fehlermeldungen gegen die Eingabe graphischer Symbole. Sie können also direkt auf dem Bildschirm "zeichnen":



Figur 2-1. Tasteneingabe in direktem Dialog

ANWEISUNGEN IN DIREKTEM DIALOG

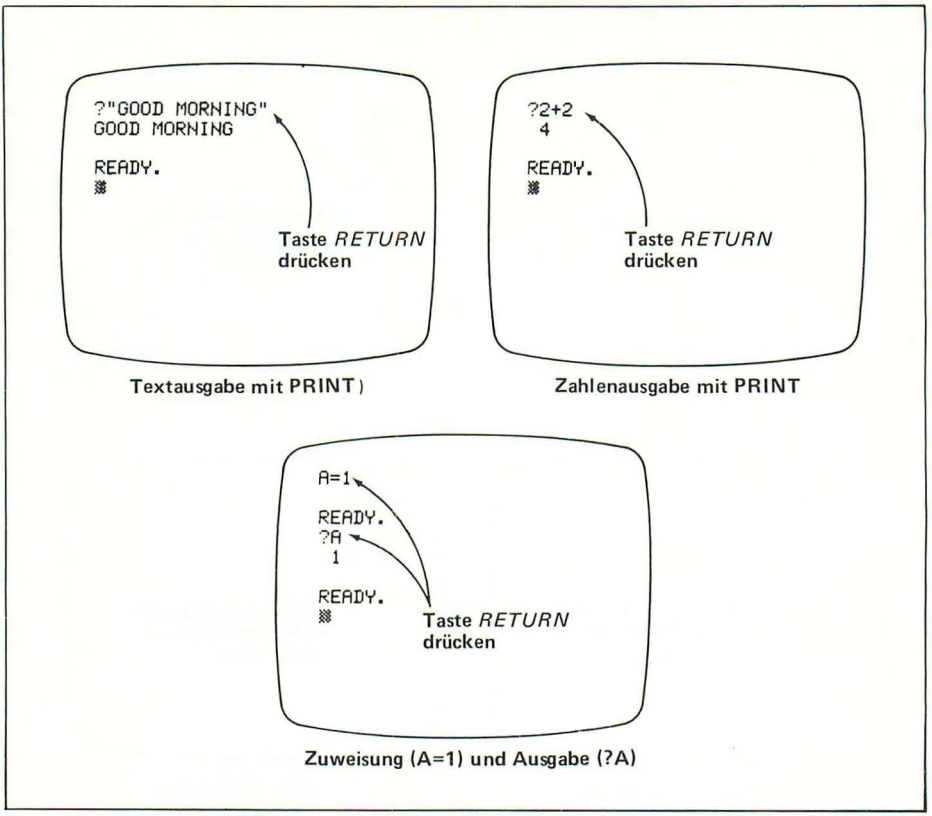
Anweisungen müssen bei direktem Dialog ohne vorangestellte Zeilennummer eingegeben werden. Wir werden später beschreiben, was alles in einer derartigen Anweisung stehen kann. Für den Augenblick wollen wir die wichtigsten Anweisungen betrachten: Anweisungen für Bildschirmausgaben, Anweisungen zur Lösung arithmetischer Aufgaben und Anweisungen zur Kursor-Bewegung.

ANWEISUNG ZUR BILDSCHIRMAUSGABE: PRINT

Die PRINT-Anweisung, abgekürzt geschrieben als ?, ist die am häufigsten benutzte Anweisung und dient zur Darstellung von Daten auf dem Bildschirm:

PRINT Daten
oder: ? Daten

Nach Drücken der Taste RETURN erscheinen die in "Daten" gemachten Angaben auf der folgenden Bildschirmzeile.



Figur 2-2. Zulässige Eingaben in direktem Dialog

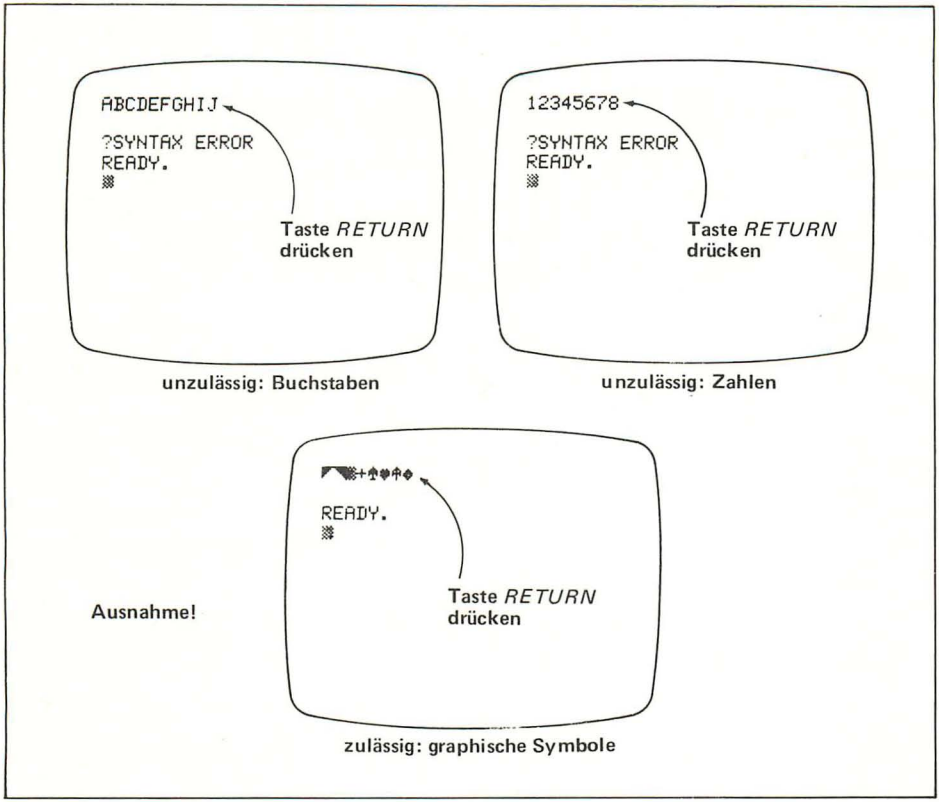
AUSGABE VON ZEICHENFOLGEN

CBM-Computer lassen die Eingabe beliebiger Zeichenfolgen zu, auch wenn diese keine Bedeutung in der Sprache CBM BASIC haben. In diesem Buch werden wir dann von "Zeichenketten" sprechen (es sind auch Begriffe wie "String", "Textfeld" und "alphanumerische Zeichenfolge" in Gebrauch).

Zur Kennzeichnung für den Computer müssen Zeichenketten in Anführungszeichen eingeschlossen sein:

```
"HI!"
"SYNERGY"
"12345"
"10.44 IS THE AMOUNT"
"22 UNION SQUARE, SAN FRANCISCO, CA"
```

Zwischen den Anführungszeichen einer Zeichenkette dürfen alle Zeichen des Tastenfeldes stehen: Buchstaben, Zahlen, Sonderzeichen und graphische Symbole. Außerdem führt die Eingabe von Kontrolltasten des Kursors zu Symboldarstellungen in einer Zeichenkette. Tab. 2-1 gibt eine Zusammenstellung der hierbei erscheinenden Symbole.



Figur 2-3. Unzulässige Eingaben in direktem Dialog

Zur direkten Ausgabe einer Zeichenkette auf dem Bildschirm verwenden Sie PRINT oder (?) in folgender Form:

PRINT "Zeichenkette"

mit:

Zeichenkette beliebige Zeichenfolge

Nach Eingabe von RETURN wird die "Zeichenkette" auf der nächsten Bildschirmzeile dargestellt. Hier einige Beispiele:

```
PRINT "MONTAG"
MONTAG

READY.

PRINT "1.APRIL 1981"
1.APRIL 1981

READY.

PRINT "12345"
12345

READY.
```

Eine Zeichenkette wird indirekt dargestellt, wenn sie nicht unmittelbar nach einer PRINT-Anweisung steht, sondern von einer "Zeichenketten-Variablen" vertreten wird. Variable (oder auch Namen) für Zeichenketten haben folgendes Aussehen:

```
A$
M1$
F6$
```

Eine Zeichenketten-Variable wird aus einem Buchstaben am Anfang und einem Dollarzeichen $ am Ende des Namens gebildet. Dazwischen kann ein beliebiges Zeichen stehen. Und so wird eine "Zeichenkette" einer Variablen zugewiesen:

Zuweisung: V$ = "Zeichenkette"

mit: V$ Variablenname für Zeichenketten

In einer PRINT –Anweisung kann jetzt statt der "Zeichenkette" stellvertretend die Zeichenkettenvariable V$ angegeben werden.

PRINT V$

mit:

V$ Variable mit zugewiesener Zeichenkette

Hier einige Beispiele für die indirekte Ausgabe von Zeichenketten:

```
A$= "NIMM DAS PROGRAMM UND LAUFE!"

READY.
PRINT A$
NIMM DAS PROGRAMM UND LAUFE!

READY.
DY$= "DIENSTAG"

READY.
PRINT DY$
DIENSTAG

READY.
B1$= "♠ ♣ ♥ O ♦"

READY.
PRINT B1$
♠ ♣ ♥ O ♦

READY.
```

Machen Sie einen eigenen Versuch mit der Eingabe Ihres Namens:

```
NM$="JIMMY OLSON ←— RETURN - Taste gedrückt

READY.
```

Wenn Sie die Fehlermeldung ?TYPE MISMATCH ERROR erhalten, haben Sie Ihren Namen entweder nicht in Anführungszeichen eingeschlossen oder den Namen der Variablen falsch angegeben. Vielleicht fehlt das Dollarzeichen. Geben Sie jetzt PRINT, gefolgt von dem Variablennamen, ein.

```
NM$="JIMMY OLSON"◄—— RETURN -Taste gedrückt

READY.
PRINT NM$ ◄———————— RETURN Taste gedrückt
JIMMY OLSON

READY.
```

Erscheint Ihr Name nicht, dann liegt ein Fehler in der PRINT-Anweisung vor. Bei einer Fehlermeldung ?SYNTAX ERROR haben Sie vielleicht PRINT falsch eingegeben. Fehlte das Dollarzeichen im Variablennamen, dann wird eine 0 ausgegeben. Stimmen die Variablennamen in der Zuweisung und in PRINT nicht überein, dann gibt es überhaupt keine Ausgabe.

ARITHMETISCHE BERECHNUNGEN

In direktem Dialog ermöglicht CBM BASIC arithmetische Operationen, die über die Möglichkeiten eines Taschenrechners hinausgehen. Im folgenden werden wir solch einfache Operationen wie Addition, Subtraktion, Multiplikation und Division besprechen und schwierigere arithmetische Operationen in Kapitel 4 besprechen. Eine arithmetische Gleichung nach einer PRINT-Anweisung weist den Computer an, die Gleichung zu lösen und das Ergebnis auf dem Bildschirm darzustellen.

Zur Lösung und Ergebnisdarstellung wird eine arithmetische Gleichung ohne Gleichheitszeichen nach einer PRINT-Anweisung angegeben:

PRINT Arithmetische Gleichung
? Arithmetische Gleichung

Hier Beispiele arithmetischer Gleichungen:

```
PRINT 2 + 2
PRINT 5/10
? 2.5
? (100/20) – 16.334
```

Nach Eingabe der RETURN-Taste löst ein CBM-Computer die angegebenen arithmetischen Gleichungen und stellt das Ergebnis auf dem Bildschirm dar:

```
PRINT 2+2 ◄———————— RETURN - Taste gedrückt
 4

READY.
```

Hier weitere Beispiele aus einem direkten Dialog mit dem Computer:

```
PRINT 2+2
 4

READY.
PRINT 5/10
 .5

READY.
? 2.5
 2.5

READY.
? (100/20)+16.334
 21.334

READY.
```

ARITHMETISCHE BERECHNUNGEN MIT VARIABLEN

Zahlen können wie Zeichenketten stellvertretend durch eine Variable dargestellt werden. Namen numerischer Variablen bestehen aus einem Buchstaben, dem ein weiterer Buchstabe oder eine Zahl folgen kann. Numerische Variable werden nicht mit einem Dollarzeichen abgeschlossen.

Zur Darstellung einer Zahl mittels einer numerischen Variablen sind zwei Anweisungen erforderlich. Mit der ersten Anweisung werden Variable und Zahl einander zugewiesen.

Zuweisung: V = n

mit: V Variablenname für Zahlen

n zugewiesene Zahl

Die zweite Anweisung stellt die der Variablen zugewiesene Zahl dar:

PRINT V

mit:

V Variable mit zugewiesener Zahl

Hier einige Zahlenbeispiele für Zuweisungen:

```
A = 1
NM = 2.56
B1 = 1000/10
```

Beachten Sie: Zahlen und Gleichungen werden nicht in Anführungszeichen eingeschlossen.

Die folgenden Beispiele für Zuweisung und Bildschirmdarstellung stammen aus einem direkten Dialog:

```
C=100

PRINT C
 100

READY.
F1=1234.78

PRINT F1
 1234.78

READY.
```

BEWEGUNG DES KURSORS

In direktem Dialog mit dem Computer stehen fünf Tasten zur Bewegung des Kursors und zur Veränderung des Schirmbilds zur Verfügung. Mit den Tasten CURSOR UP/DOWN, CURSOR LEFT/RIGHT und HOME kann der Kursor über den ganzen Bildschirm bzw. zum Bildschirmanfang bewegt werden, ohne vorhandene Darstellungen zu ändern. Die Taste CLEAR SCREEN löscht alle Darstellungen auf dem Bildschirm und führt den Kursor an den Bildschirmanfang zurück. Die Taste INSERT/DELETE dient zur Korrektur (Edition) von Textdarstellungen auf dem Bildschirm: Mit INSERT entstehen rechts des Kursors Leerstellen für Texteinfügungen, DELETE

bewirkt eine Linksbewegung des Kursors, wobei überfahrener Text gelöscht wird.

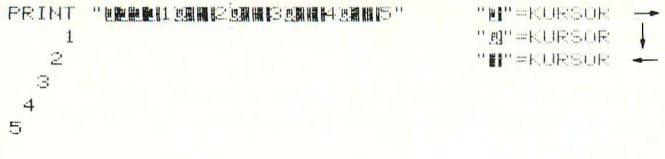
Nach einem Anführungszeichen (oder einer ungeraden Anzahl von Anführungszeichen) ändert sich die Wirkung der meisten Kursor-Tasten: Statt Ausführung der Kursorbewegung wird ein Symbol dargestellt. Kursortasten werden nach vorangehendem Anführungszeichen als Zeichen einer Zeichenkette dargestellt. Tabelle 2-1 zeigt die erscheinenden Symbole.

Tabelle 2-1. Darstellung von Funktionstasten in Zeichenketten

Funktion	Tasteneingabe	Graph. Symbol
Text-löschung	INST DEL	nicht programmierbar
Text-einfügung	SHIFT + INST DEL	(SHIFT + RVS + T)
Kursor zum Bildschirmanfang	CLR SCREEN HOME	(RVS + S)
Bildschirm-löschung	SHIFT + CLR SCREEN HOME	(SHIFT + RVS + S)
Kursor ↓	CURSOR	(RVS + Q)
Kursor ↑	SHIFT + CURSOR	(SHIFT + RVS + Q)
Kursor →	CURSOR	(RVS + J)
Kursor ←	SHIFT + CURSOR	(SHIFT + RVS + J)

Es folgen Beispiele von Zeichenketten, die Symbole für programmierte Kursorbewegungen enthalten. Die jeweils unter PRINT erscheinenden Darstellungen lassen die Wirkung der programmierten Kursorbewegungen erkennen:

```
PRINT "1■2■3■4■5"                    "■"=KURSOR →
1 2 3 4 5

READY.

PRINT "1■2■3■4■5"                    "■"=KURSOR ←
1                                    "■"=KURSOR ↓
2
3
4
5

READY.
```



PROGRAMMFORM

Als Programm eingegebene Anweisungen werden im Computer gespeichert. Derartige Anweisungen müssen mit einer Zeilennummer beginnen. Ein Programm besteht aus einer oder mehreren Anweisungen in CBM BASIC, die der Reihe nach im Computer ausgeführt werden und hierbei eine gewünschte Aufgabe lösen.

Programme können über das Tastenfeld eingegeben werden. Programme können aber auch von einem externen Speicher wie Bandkassette oder Floppy-Disk in den Speicher des Computers geladen werden.

Bei der Eingabe am Tastenfeld wird jeder Anweisung eine Zeilennummer vorangesetzt. Wird am Ende der Eingabe einer Anweisung die Taste RETURN gedrückt, dann speichert der Computer diese Anweisung für eine spätere Benutzung. Die eingegebene Anweisung bleibt für den Benutzer auf dem Bildschirm sichtbar. Ein Programm bleibt im Computer, bis der Speicher des Computers gelöscht oder der Computer ausgeschaltet wird. Um den Verlust eines Programms zu vermeiden, können Sie es entweder auf der Kassette eines Bandgerätes oder auf der Diskette eines Floppy-Disk-Speichers speichern. Sobald ein Programm auf eins dieser Speichermedien übertragen wurde, kann es zu jeder Zeit in den Speicher des Computers zurückgeholt werden.

Sie werden jetzt erfahren, wie man ein CBM BASIC-Programm schreibt; in aller Ausführlichkeit wird das Schreiben von Programmen in Kapitel 4 behandelt. Bis jetzt wissen Sie aber bereits genug, um das folgende kurze Programm mit dem Namen BLANKET in Ihren Computer einzugeben:

```
10 FOR I=1 TO 800
20 PRINT "A";
30 NEXT I
40 PRINT "PHEW!"
50 END
```

Seien Sie nicht beunruhigt, wenn Sie im Augenblick CBM BASIC oder dieses Programm BLANKET nicht verstehen. Wir werden dieses Programm benutzen, um die Arbeitsweise eines Computers zu zeigen, und hierbei werden Sie mit BASIC und dem Programm BLANKET vertraut werden.

Wenn Sie bei der Programmeingabe einen Fehler machen, drücken Sie die RETURN-Taste und wiederholen die Eingabe. Auch nach einer Fehlermeldung ?SYNTAX ERROR wiederholen Sie ebenfalls die Eingabe.

Zuerst wollen wir uns ansehen, wie ein im Computer gespeichertes Programm auf dem Bildschirm sichtbar zu machen ist.

PROGRAMMAUFLISTUNG: LIST

Die Anweisung LIST bringt alle augenblicklich im Computer gespeicherten Programme auf dem Bildschirm zur Darstellung. Ein auf dem Bildschirm aufgelistetes Programm kann auf Fehler untersucht oder geändert werden. Entnehmen Sie das Format einer LIST-Anweisung den folgenden Beispielen:

```
LIST          ALLE GESPEICHERTEN PROGRAMME AUFLISTEN
LIST 50       PROGRAMMZEILE MIT ZEILENNUMMER 50 AUFLISTEN
LIST 10-100   ALLE PROGRAMMZEILEN MIT ZEILENNUMMERN 10 BIS 100 AUFLISTEN
LIST 10-      ALLE PROGRAMMZEILEN AB  ZEILENNUMMER 10 AUFLISTEN
LIST -200     ALLE PROGRAMMZEILEN BIS ZEILENNUMMER 200 AUFLISTEN

LIST XX-YY    ZEILENNUMMER XX IST KLEINER ALS YY. AUFLISTUNG
              EINSCHLIESS XX,YY. ZEILENNUMMERN XX,YY MUESSEN NICHT
              UNBEDINGT EXISTIEREN.
LIST XX       BEI NICHT EXISTIERENDER ZEILENNUMMER XX WIRD LEERZEILE
              GEDRUCKT.
```

Ist ein Programm länger als 23 Programmzeilen, dann werden nur die letzten 23 Programmzeilen auf dem Bildschirm dargestellt, während der Programmanfang nach oben aus dem Bildschirm wandert. Benutzen Sie in solchen Fällen die LIST-Anweisung unter Angabe von Zeilennummern.

Machen Sie jetzt einen Versuch mit dem eben eingegebenen Programm BLANKET. Geben Sie die Buchstaben LIST ein und drücken danach die Taste RETURN. Auf dem Bildschirm sollte Ihr Programm erscheinen:

```
LIST

 10 FOR I=1 TO 800
 20 PRINT "A";
 30 NEXT I
 40 PRINT "PHEW!"
 50 END
READY.
```

Beachten Sie, daß während der Auflistung des Programms auf dem Bildschirm der Kursor verschwindet. Am Ende der Auflistung erscheint der Kursor wieder unterhalb des Programms, um auf weitere Eingaben in direktem Dialog zu warten.

Üben Sie jetzt die Auflistung eines Teils des Programms. Die Auflistung der Programmzeile 10 ergibt sich bei folgender Eingabe:

```
LIST 10

 10 FOR I=1 TO 800

READY.
```

Die Zeilennummern 20 bis 40 erscheinen nach folgender Eingabe:

```
LIST 20-40

 20 PRINT "A";
 30 NEXT I
 40 PRINT "PHEW!"
READY.
```

Machen Sie auf diese Weise mit allen Formen von LIST Versuche.

PROGRAMMSTART: RUN

Die Anweisung RUN läßt ein augenblicklich im Computer gespeichertes Programm ablaufen. Sobald Sie RUN eingegeben haben und die RETURN-Taste drücken, beginnt der Programmablauf. Haben Sie mehrere Programme im Speicher des Computers, dann können Sie wählen, welches der Programme Sie ablaufen lassen wollen. Sie geben einfach nach RUN die Zeilennummer der ersten Programmzeile des gewünschten Programms an:

RUN Programmablauf beginnt bei kleinster Zeilennummer

RUN xx Programmablauf beginnt bei Zeilennummer xx

Bringen Sie jetzt das Programmbeispiel zum Ablauf. Der Bildschirm sollte mit 800 A's beschrieben werden. Beachten Sie, daß während der Programmausführung der Kursor verschwindet und nach Programmende zusammen mit der Nachricht READY am unteren Bildschirmende wieder erscheint. Der Computer ist jetzt wieder bereit für einen direkten Dialog:

```
RUN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
PHEW!

READY.
▒
```

Mit einer Anweisung RUN xx können Sie auch den Programmablauf mit jeder Programmzeile beginnen, wenn Sie die Zeilennummer xx angeben. Lassen Sie in unserem Programmbeispiel den Ablauf bei Programmzeile 40 beginnen, wie folgendes Beispiel zeigt:

```
RUN 40
PHEW!

READY.
▒
```

Sie sollten vorsichtig sein, wenn Sie mit RUN einen Programmablauf mitten im Programm beginnen lassen. Häufig läuft ein Programm nur dann richtig ab, wenn der Ablauf am Programmanfang beginnt. Im Programmbeispiel wäre ein Ablauf von Zeile 20 oder 30 nicht möglich, der Computer würde eine Fehlermeldung abgeben.

BEENDEN DES PROGRAMMABLAUFS: STOP

Der Ablauf eines Programms kann unterbrochen werden. Drücken Sie während des Programmablaufs die Taste STOP. Im Anschluß an eine Unterbrechung erscheint auf dem Bildschirm eine Nachricht, aus der die Zeilennummer hervorgeht, an der die Unterbrechung erfolgte:

```
RUN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA ←———————————————— STOP –Taste gedrückt
BREAK IN 40 ←——————————————————— Unterbrechungsmeldung
READY.
▒
```

Nach einer Programmunterbrechung kehrt ein CBM-Computer zum direkten Dialog zurück, so daß Sie Gelegenheit für eine Programmauflistung mit LIST zur Ausführung von Programmänderungen haben, oder eine andere Operation ausführen können, ehe Sie den Programmablauf fortsetzen.

FORTSETZUNG DES PROGRAMMABLAUFS: CONT

Mit der Eingabe CONT kann ein zuvor unterbrochener Programmablauf fortgesetzt werden. Die Fortsetzung erfolgt genau dort, wo die Unterbrechung erfolgte:

```
RUN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA ←————————————————— STOP -Taste gedrückt
BREAK IN 40
READY.
CONT ←—————————————————————————— Eingabe von CONT und RETURN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAPHEW!

READY.
▒
```

Mit CONT kann ein Programmablauf dann nicht fortgesetzt werden, wenn ein Ablauffehler vorlag und auf dem Bildschirm eine Fehlermeldung erschienen ist.

PROGRAMMLÖSCHUNG: NEW

Sie können alle im Speicher Ihres Computers gespeicherten Programme löschen. Zur Löschung geben Sie NEW ein und drücken die Taste RETURN. Hier ein Beispiel:

```
LIST ◄———————— alle eingegebenen Programme auflisten

 10 FOR I=1 TO 800
 20 PRINT "A";
 30 NEXT I              aufgelistetes Programm (Beispiel)
 40 PRINT "PHEW!"
 50 END
READY.
NEW ◄———————— alle eingegebenen Programme löschen

READY.
LIST ◄———————— alle eingegebenen Programme auflisten
     ◄———————— keine Programmauflistung
READY.
■
```

Löschen Sie jeweils vor Eingabe eines neuen Programms ein im Speicher vorhandenes altes Programm. Sonst behalten Sie im Speicher ein Potpourri alter und neuer Programmanweisungen.

VERFÜGBARE ZEICHENSÄTZE

Am Tastenfeld jedes CBM-Computers stehen zwei Zeichensätze zur Verfügung, die in Kapitel 1 und in Anhang A beschrieben sind.

Der Standard-Zeichensatz besteht aus Großbuchstaben und graphischen Symbolen. Der Alternativ-Zeichensatz besteht aus Kleinbuchstaben und Großbuchstaben. Bei Inbetriebnahme der Modelle CBM 8000 steht am Tastenfeld der alternative Zeichensatz zur Verfügung.

Zum Übergang auf einen anderen Zeichensatz geben Sie folgenden Befehl ein:

POKE 59468,12 **Standard-Zeichensatz**

POKE 59468,14 **Alternativer Zeichensatz**

PROGRAMMIERUNG IM ALTERNATIVEN ZEICHENSATZ

Beim Arbeiten im alternativen Zeichensatz dürfen Sie niemals zur Eingabe von Anweisungen in CBM BASIC die SHIFT-Taste benutzen, um auf Großbuchstaben umzuschalten.

Alle CBM-Computer nehmen an, daß eine Programmeingabe im Standard-Zeichensatz erfolgt. Wenn Sie zur Eingabe den alternativen Zeichensatz verwenden, dann dürfen Sie nicht die SHIFT-Taste benutzen: Der Computer erwartet nach SHIFT graphische Symbole, und nicht die Großbuchstaben des alternativen Zeichensatzes.

Geben Sie also nicht Anweisungen mit den Möglichkeiten des alternativen Zeichensatzes wie folgt ein:

```
10 For I=1 To 10 Step 2
```

Alle Großbuchstaben in diesem Beispiel werden vom CBM-Computer als graphische Symbole verstanden, wie sie der Standard-Zeichensatz bei SHIFT erzeugt:

```
10 FOR I=1 TO 10 STEP 2
```

Der CBM-Computer würde diese Anweisung ablehnen und eine Fehlermeldung ausgeben. Innerhalb der Anführungszeichen einer Zeichenkette kann der alternative Zeichensatz ohne Einschränkung verwendet werden.

ARBEITEN MIT DEM BANDGERÄT

Kleinere CBM-Computersysteme machen von Bandkassetten-Speichern Gebrauch, um Programme und Daten zu speichern. Größere CBM-Computer-Systeme verwenden Floppy-Disk-Speicher für den gleichen Zweck. Mit den folgenden Anweisungen zeigen wir Ihnen den Umgang mit CBM-Bandgeräten.

Vergewissern Sie sich zunächst, daß die richtige Kassette in das Bandgerät eingelegt wurde. Drücken Sie die Taste REWIND zum Zurückspulen des Bandes. Alle Tasten am Bandgerät sollen in Aufwärtsstellung stehen.

Das Arbeiten mit einem Bandgerät werden wir am Beispiel des Programms BLANKET zeigen. In den angegebenen Beispielen sind Ihre Eingaben durch Schattierung gekennzeichnet, während die Antworten des Computers unschattiert wiedergegeben sind.

PROGRAMMSPEICHERUNG: SAVE

Die Anweisung SAVE schreibt ein augenblicklich im Speicher des Computers stehendes Programm auf eine Bandkassette:

SAVE "Programmname"		**Bandgerät #1**
SAVE "Programmname", 1	**Programmspeicherung auf**	**Bandgerät #1**
SAVE "Programmname", 2		**Bandgerät #2**

Fehlt die Angabe einer Gerätenummer, dann erfolgt die Speicherung automatisch in Bandgerät # 1. Lediglich das Arbeiten mit Bandgerät # 2 muß explizit angegeben werden.

Speichern Sie zur Übung das Programm BLANKET auf einer Bandkassette. Laden Sie eine leere Kassette in das Bandgerät # 1. Geben Sie das Programm über das Tastenfeld ein. Speichern Sie dann dieses Programm unter dem Namen BLANKET mit einer SAVE-Anweisung wie folgt:

```
SAVE "BLANKET"

PRESS PLAY & RECORD ON TAPE #1
```

Nach Drücken der Taste RETURN erscheint die angegebene Meldung des Computers, die besagt: Drücken Sie die Tasten PLAY und RECORD am Bandgerät

1. Während des folgenden Bandvorlaufs wird das Programm BLANKET aufgezeichnet. Achten Sie darauf, daß wirklich beide Tasten, PLAY und RECORD, gedrückt wurden, da sonst Sie und Ihr CBM-Computer glauben, daß das Programm aufgezeichnet wird, was aber nicht geschieht. Während der Dauer der Aufzeichnung erscheint folgende Mitteilung an den Benutzer:

```
SAVE "BLANKET"

PRESS PLAY & RECORD ON TAPE #1
OK
WRITING BLANKET
READY.
```

Mitteilung an den Benutzer: Das Programm BLANKET wird aufgezeichnet

Nach Abschluß der Aufzeichnung kehrt der Computer mit der Meldung READY zum direkten Dialog zurück.

Warnung: Die Anweisung SAVE für eine Bandaufzeichnung wird ausgeführt, sobald irgendeine der Tasten REC, REW, FFWD oder PLAY am Bandgerät gedrückt wurde. Bedienen Sie vor Ausführung einer SAVE-Anweisung die Taste STOP am Bandgerät; hierdurch wird sichergestellt, daß die Aufforderung PRESS PLAY & RECORD ON TAPE # x erfolgt. Achten Sie darauf, die Tasten PLAY und RECORD gleichzeitig zu drücken.

Nach Speicherung eines Programms mit SAVE können Sie den Erfolg der Bandaufzeichnung mit VERIFY prüfen.

PRÜFUNG EINER BANDAUFZEICHNUNG: VERIFY

Die Anweisung VERIFY stellt Aufzeichnungsfehler bei der Speicherung eines Programms auf Bandkassette fest. Hierfür simuliert VERIFY das Laden eines Programms in den Computer, wobei es das geladene und das noch im Speicher vorhandene Programm vergleicht und im Fall einer Abweichung eine Mitteilung auf dem Bildschirm ausgibt.

Für VERIFY gilt folgende Syntax:

VERIFY	**Prüfung einer gerade erfolgten Programmaufzeichnung in Bandgerät 1**	
VERIFY "Programmname"	**Prüfung der Programmaufzeichnung in**	**Bandgerät 1**
VERIFY "Programmname", 1		**Bandgerät 1**
VERIFY "Programmname", 2		**Bandgerät 2**

Gehen Sie für die Prüfung einer Programmaufzeichnung in folgenden Schritten vor:

1. Spulen Sie das Band an den Anfang zurück oder bis zu einer Stelle, die vor der gerade erfolgten Programmaufzeichnung liegt.
2. Drücken Sie die STOP-Taste.
3. Geben Sie die VERIFY-Anweisung an.

Nach einer gerade erfolgten Programmaufzeichnung im Bandgerät # 1 sind keine weiteren Angaben in VERIFY erforderlich.

Für das Programm BLANKET nehmen Speicherung und Prüfung der Aufzeichnung folgende Form an:

```
SAVE "BLANKET"

PRESS PLAY & RECORD ON TAPE #1 <-------- Tasten PLAY und RECORD an
OK                                       Bandgerät #1 drücken

WRITING BLANKET
READY.
VERIFY "BLANKET"

PRESS PLAY ON TAPE #1 <---------------- Taste PLAY an Bandgerät #1
OK                                      drücken; Quittung: OK

SEARCHING FOR BLANKET
FOUND BLANKET
VERIFYING
OK

READY.
```

Die Prüfung der Programmaufzeichnung beginnt nach Drücken der Taste PLAY. Während der Suche nach dem Programm BLANKET auf der Bandkassette erscheint die Mitteilung SEARCH FOR BLANKET. Die Nachricht FOUND BLANKET teilt mit, daß das Programm gefunden wurde. Prüfung und positives Ergebnis der Prüfung werden durch die Mitteilungen VERIFY und OK mitgeteilt.

Die Nachricht ?VERIFY ERROR erscheint, wenn ein Aufzeichnungsfehler entdeckt wurde. Spulen Sie in diesem Fall das Band zurück und wiederholen Sie die Prüfung. Verschwindet die Fehlermeldung nicht, dann spulen Sie das Band zum Anfang der Programmaufzeichnung zurück und wiederholen die Programmspeicherung mit SAVE. Erscheint die Fehlermeldung wieder, dann wechseln Sie die Kassette.

LADEN EINES PROGRAMMS: LOAD

Die Anweisung LOAD lädt ein Programm vom Bandgerät in den Speicher des Computers. Zum Laden eines Programms vom Bandgerät # 1 verwenden Sie eine der folgenden Anweisungen:

LOAD "Programmname"

LOAD "Programmname", 1

Fehlt ein Programmname in der Anweisung LOAD, dann wird das nächste auf der Kassette angetroffene Programm in den Computer geladen.

Fehlt eine Gerätenummer nach dem Programmnamen, dann wird automatisch Bandgerät # 1 angesprochen. Je nach Modell des CBM-Computers ist Bandgerät # 1 entweder das eingebaute Bandgerät neben dem Tastenfeld oder das über den Stecker J3 angeschlossene Bandgerät. Ein wahlweise anzuschließendes weiteres Bandgerät trägt die Gerätenummer 2.

Wenn Sie nur mit Bandgerät # 1 arbeiten, entfällt die Angabe einer Gerätenummer. Zum Laden des Programms BLANKET genügt daher eine der folgenden LOAD-Anweisungen:

```
    LOAD "BLANKET"

oder: LOAD "BLANKET",1
```

Das folgende Beispiel zeigt den vollständigen Ablauf des Ladevorgangs:

```
LOAD "BLANKET"

PRESS PLAY ON TAPE #1
OK

SEARCHING FOR BLANKET
FOUND BLANKET
LOADING
READY.
```

Taste PLAY an Bandgerät #1 drücken

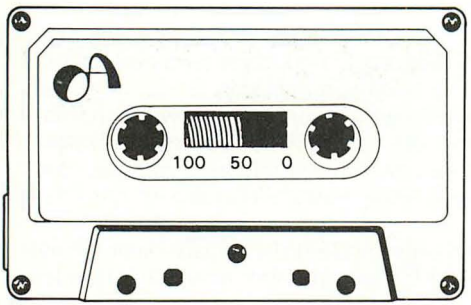
Nach Eingabe von LOAD prüft der Computer, ob die Taste PLAY am Bandgerät gedrückt ist. Wenn nicht, erscheint eine Aufforderung an den Benutzer. Während der Suche nach dem gewünschten Programm erscheint die Nachricht SEARCHING FOR . . . auf dem Bildschirm. Jedes auf der Kassette gefundene Programm wird nach dem Wort FOUND mitgeteilt, bis das gewünschte Programm gefunden wurde. Während des Ladens dieses Programms erscheint die Nachricht LOADING und nach Abschluß des Ladens die Rückkehr zum direkten Dialog mit der Nachricht READY.

Sollen weitere Programme von derselben Kassette geladen werden, dann können Sie die Taste PLAY niedergedrückt lassen. Soll jedoch später eine Programmspeicherung erfolgen, dann müssen Sie die Taste PLAY in Aufwärtsstellung bringen. Die Taste PLAY verursacht die meisten Bedienungsfehler bei abwechselndem LOAD und SAVE von Programmen.

Manchmal ergeben sich Schwierigkeiten beim Laden eines Programms vom Bandgerät in den Speicher des Computers. Der CBM-Computer findet zwar den Programmnamen, lädt aber nicht das Programm richtig oder lädt es überhaupt nicht. Wiederholen Sie in solchen Fällen den Ladevorgang. Wenn das nicht hilft, können Fehler bei der Speicherung oder im Bandgerät der Grund sein.

Vergewissern Sie sich vor Auslösen einer LOAD-Anweisung, daß das Magnetband bis zu einer Stelle vor der erfolgten Programmaufzeichnung zurückgespult wurde. Sonst sucht der Computer das Magnetband bis zu Ende ab, ohne das Programm zu finden. Zur Verkürzung der Suchzeit nach einem Programm, das am Ende des Magnetbands steht, können Sie das Band mit der Taste FFWD bis zu einer geeigneteren Stelle vorlaufen lassen. Die LOAD-Anweisung wird dann schneller erledigt.

Ein Verfahren zur groben Lokalisierung von Programmen auf der Kassette besteht in der Benutzung der Meßskala für die Bandlänge, wie sie folgendes Bild zeigt.



Während einer Bandbewegung ändert sich der Spulenradius, der an der Skala abgelesen werden kann. Sie können sich vor Beginn einer Programmaufzeichnung die Skalenablesung notieren, und vor dem Laden des Programms das Band wieder an etwa die gleiche Stelle bringen.

Zur Vergewisserung können Sie ein gerade geladenes Programm mit der Anweisung LIST auf dem Bildschirm darstellen.

Das vom Bandgerät geladene Programm wird dann durch Eingabe von RUN zum Ablauf gebracht.

LADEN UND ABLAUFEN: LOAD, RUN

BASIC < 3.0

Mit der Taste LOAD & RUN wird automatisch ein Programm von Bandgerät # 1 geladen und zum Ablauf gebracht. LOAD & RUN entsteht durch Drücken der Tasten SHIFT und RUN/STOP. LOAD & RUN ist nur auf CBM-Computern mit BASIC < 3.0 verfügbar.

Da mit LOAD & RUN keine Angabe eines Programmnamens verbunden ist, kann jeweils nur ein Programm geladen und zum Ablauf gebracht werden. Positionieren Sie das Magnetband in Bandgerät # 1 auf eine Stelle vor der gewünschten Programmaufzeichnung. Der Computer lädt automatisch das nächste angetroffene Programm auf der Kassette und läßt es ablaufen:

```
LOAD                                          <-- Tasten SHIFT und RUN/STOP drücken

PRESS PLAY ON TAPE #1                         <-- Taste PLAY an Bandgerät # 1
OK                                                drücken

FOUND BLANKET
LOADING
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA    <-- Automatischer Programm-
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA        ablauf
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
.....
```

Die Tasten SHIFT und RUN können nur in direktem Dialog eingegeben werden. Werden Sie während eines Programmablaufs bedient, dann wirken sie wie ein STOP und unterbrechen den Programmablauf.

ARBEITEN MIT DEM FLOPPY-DISK-SPEICHER

Im folgenden schildern wir, wie Programme auf einer Diskette gespeichert und von einer Diskette in den Computer geladen werden.

Kenntnisse in der Programmierung mit CBM BASIC sind nicht erforderlich. Ausführliche Beschreibungen aller mit Floppy-Disk-Speichern möglichen Arbeiten finden Sie in Kapitel 6.

Am Ende dieses Abschnitts werden Sie in der Lage sein, beide Laufwerke Ihres Floppy-Disk-Speichers zu initiieren, Programme auf einer Diskette zu speichern, Programme von einer Diskette zu laden und zum Ablauf zu bringen und sich ein Inhaltsverzeichnis aller auf Disketten gespeicherten Programme ausgeben zu lassen.

BETRIEBSSYSTEME FÜR FLOPPY-DISK-SPEICHER

Ihr CBM-Computer ist mit Programmen ausgerüstet, die den Datenverkehr mit einem Floppy-Disk-Speicher abwickeln können. Diese Programme werden zusammenfassend "Disketten-Betriebs-System" (DOS im Englischen) genannt.

Als Benutzer eines Floppy-Disk-Speichers brauchen Sie diese Programme nicht zu kennen.

Gegenwärtig sind mehrere verschiedene Ausgaben des CBM-Betriebssystems für Disketten in Gebrauch. Da ein Betriebssystem das Spektrum von Möglichkeiten für die Benutzung von Floppy-Disk-Speichern bestimmt, ist für Sie von Interesse, welcher CBM-Computer mit welchem Betriebssystem für Disketten arbeiten kann:

	CBM DOS 1.X CBM BASIC <3.0	CBM DOS 2.X CBM BASIC 4.0
FLOPPY DISK SPEICHER	CBM 3040	CBM 4040 CBM 8050
	CBM BASIC <3.0	CBM BASIC 4.0
COMPUTER SERIE	CBM 2000 CBM 3000 *	CBM 4000 CBM 8000

* NACHRUESTBAR AUF BASIC 4.0

LADEN EINES PROGRAMMS VON DISKETTE IN BASIC < 3.0

Ein Programm wird in folgenden Schritten von einer Diskette geladen:

1. Öffnen einer logischen Datei und Angabe der Gerätenummer
2. Initiieren eines Laufwerks
3. Darstellen des Inhaltsverzeichnisses einer Diskette
4. Laden des Programms

Zur Veranschaulichung dieser Schritte verwenden wir die jedem Floppy-Disk-Speicher beigefügte TEST/DEMO-Diskette.

Wir fangen an mit dem Laden der TEST/DEMO-Diskette in Laufwerk 0, entsprechend den Anweisungen in Kapitel 1.

ÖFFNEN EINER LOGISCHEN DATEI: OPEN — BASIC < 3.0

Vor dem Laden eines Programms von einer Diskette muß zunächst ein Datenkanal vom Computer zum Floppy-Disk-Speicher geöffnet und für den Datenverkehr vorbereitet werden. Hierfür geben wir in direktem Dialog eine OPEN-Anweisung ein:

```
OPEN 1,8,15
```

INITIIEREN EINES LAUFWERKS: INITIALIZE — BASIC < 3.0

In BASIC < 3.0 muß ein Laufwerk jedesmal initiiert werden, wenn eine Diskette in das Laufwerk eingesetzt wird oder wenn Disketten zwischen den Laufwerken

ausgetauscht wurden. Ein Laufwerk muß eine Diskette enthalten, ehe es initiiert werden kann.

Zur Initiierung dient die Anweisung INITIALIZE, abgekürzt "I":

PRINT #Datei, "INITIALIZE lfw"

mit:

Datei	logische Dateinummer wie in OPEN
INITIALIZE	abgekürzt: I
lfw	Laufwerknummer 0 oder 1

Laufwerk Null mit der TEST/DEMO-Diskette wird wie folgt initiiert:

```
PRINT#1,"I0"
```

Während der Initiierung arbeitet der Motor in Laufwerk 0 und die Anzeige-Diode leuchtet auf.

Zur Initiierung von Laufwerk 1 laden Sie eine Diskette in Laufwerk 1 und machen folgende Eingabe:

```
PRINT#1,"I1"
```

Die Initiierung beider Laufwerke wird von folgender Anweisung ausgelöst:

```
PRINT#1,"I"
```

DARSTELLEN DES INHALTSVERZEICHNISSES: LOAD BASIC < 3.0

Wenn Sie schon den Namen des zu ladenden Programms kennen, können Sie diesen Schritt überspringen.

Das Inhaltsverzeichnis einer Diskette dient dem gleichen Zweck wie das Inhaltsverzeichnis eines Buches: Es führt die Namen der auf der Diskette gespeicherten Programme auf.

Das Inhaltsverzeichnis für die Diskette in Laufwerk 0 erhalten Sie mit:

```
LOAD "$0",8
```

Für das Inhaltsverzeichnis der Diskette in Laufwerk 1 machen Sie folgende Eingabe:

```
LOAD "$1",8
```

Bei fehlender Laufwerknummer werden die Inhaltsverzeichnisse beider Disketten ausgegeben:

```
LOAD "$",8
```

Während des Ladens eines Inhaltsverzeichnisses werden folgende Nachrichten auf dem Bildschirm dargestellt:

```
LOAD"$0",8

SEARCHING FOR $0
LOADING
READY.
```

Geben Sie nach Erscheinen von READY die Anweisung LIST ein, um das Inhaltsverzeichnis auf dem Bildschirm zur Darstellung zu bringen. Ist das Inhaltsverzeichnis länger als 25 Zeilen, dann wandern die Anfangszeilen nach oben aus dem

Bildschirm. Mit der Taste STOP können Sie eine Auflistung zum Halten bringen. Die Auflistung läuft verlangsamt ab, wenn Sie die Taste REVERSE oder die Taste ← auf Geschäftstastaturen gedrückt halten.

Durch wiederholte Eingabe der LIST-Anweisung können Sie das Inhaltsverzeichnis so oft darstellen, wie Sie wünschen. Hier das Beispiel des Inhaltsverzeichnisses einer der TEST/DEMO-Disketten:

```
LIST

0 "TEST/DEMO          " 2A
6     "DOS SUPPORT 4.0"  PRG
27    "DUM 3.4"          PRG
1     "  DISK DATA  "    SEQ
15    "DIAGNOSTIC BOOT"  PRG
10    "COPY DISK FILES"  PRG
4     "CHECK DISK"       PRG
10    "PET DISK"         PRG
10    "DISK DISPLAY"     PRG
3     "DISK COMM"        PRG
2     "DISK COMM2"       PRG
3     "DISK COMM3"       PRG
4     "DISK WRITE"       PRG
4     "DISK READ"        PRG
2     "DISK OVERLAYS"    PRG
5     "DISK DIR"         PRG
7     "  PET DATA  "     SEQ
34    "RANDOM 1.00"      PRG
27    "PRINTER DEMO"     PRG
12    "SEQUENTIAL 1.00"  PRG
484 BLOCKS FREE.
```

Die erste Zeile stellt den "Kopfeintrag" dar. Der Kopfeintrag in Negativschrift zeigt den Namen der Diskette und eine Nummer zu ihrer Identifizierung. Die mittlere Spalte stellt die Auflistung aller Programmnamen dieser Diskette dar. Die Anführungszeichen sind nicht als Teil des Programmnamens zu verstehen. Die linke Spalte zeigt die Blockbelegung, d.h. den vom Programm belegten Speicherplatz. Die letzte Zeile des Inhaltsverzeichnisses gibt die Gesamtzahl noch freier Speicherblöcke auf der Diskette an. (Kapitel 6 enthält eine ausführliche Beschreibung der Speicherorganisation von Disketten). Die rechte Spalte gibt an, ob die Information als Programm, PRG, oder Daten, z.B. SEQ, gespeichert wurde. Für den Augenblick sind die linke und rechte Spalte ohne Bedeutung für Sie. Überfliegen Sie die mittlere Spalte, um ein gewünschtes Programm zu finden und zu laden.

LADEN EINES PROGRAMMS: LOAD

BASIC < 3.0

Die Anweisung LOAD wird in direktem Dialog eingegeben, um ein Programm von einer Diskette in den Speicher des Computers zu laden. Für eine Diskette in Laufwerk 0 hat LOAD folgende Form:

LOAD "0: Programmname", 8

Laufwerk-Nummer und Programmname sind durch einen Doppelpunkt getrennt; beide Angaben stehen zwischen Anführungszeichen. Wird keine Laufwerknummer 0 oder 1 angegeben, dann sucht das Disketten-Betriebssystem in beiden Laufwerken nach dem gewünschten Programm, vorausgesetzt, daß beide Laufwerke zuvor initiiert wurden.

Wir laden jetzt das erste Programm auf der TEST/DEMO-Diskette in Laufwerk 0:

```
LOAD "0:DOS SUPPORT 4.0",8
```

Nach Eingabe von RETURN erscheint folgende Meldung auf dem Bildschirm, während gleichzeitig die Anzeigelampe an Laufwerk 0 aufleuchtet und das Laufwerk einen Summton von sich gibt:

```
LOAD "0:DOS SUPPORT 4.0",8

SEARCHING FOR 0:DOS SUPPORT 4.0
LOADING
READY.
```

Die Suchmeldung SEARCHING FOR gibt die Laufwerknummer und den Programmnamen des gesuchten Programms an. Der Ladevorgang des Programms ist beendet, wenn die Anzeigelampe erlischt, der Summton verschwindet und die Nachricht READY auf dem Bildschirm erscheint. Das Programm ist jetzt im Computer gespeichert und kann aufgelistet oder zum Ablauf gebracht werden.

PROGRAMMAUFLISTUNG: LIST

BASIC < 3.0

Sobald ein Programm in den Speicher des Computers geladen wurde, kann es zur Vergewisserung mit der Anweisung LIST auf dem Bildschirm dargestellt werden.

PROGRAMMABLAUF: RUN

BASIC < 3.0

Ein gerade von einer Diskette geladenes Programm bringen Sie zum Ablauf, indem Sie RUN eingeben und danach die Taste RETURN drücken.

ANLEGEN EINER NEUEN DISKETTE: NEW

BASIC < 3.0

Eine fabrikneue Diskette kann nicht in einem Floppy-Disk-Speicher verwendet werden, ehe ihre Oberfläche nicht für die Datenspeicherung vorbereitet oder "formatiert" wurde. Wenden Sie einfach die folgenden Anweisungen an, um eine neue Diskette zu formatieren oder bei einer alten Diskette die Formatierung zu wiederholen.

Zunächst öffnen Sie den Datenverkehr zum Floppy-Disk-Speicher:

```
OPEN 1,8,15
```

Eine PRINT-Anweisung mit der Angabe NEW löst die Formatierung der Diskette und eine Initiierung des Laufwerks aus:

PRINT #Datei, "NEW lfw: Diskname, xx;"

mit:

Datei	logische Dateinummer wie in OPEN
NEW	zulässige Abkürzung: N
lfw	Laufwerknummer, 0 oder 1
Diskname	Name der Diskette
xx	2stellige Kennzahl

In unserem Beispiel befindet sich die Diskette in Laufwerk 0, daher ist Lfw=0. Wir geben unserer Diskette den Namen YAK und als Kennzahl 01. Mit diesen Angaben lautet die Formatierung der Diskette wie folgt:

```
OPEN 1,8,15 : PRINT#1,"N0:YAK,01"
```

Drücken Sie die Taste RETURN. Am Laufwerk 0 leuchtet die Anzeigelampe auf, während die Formatierung ausgeführt wird. Der Kursor verschwindet für kurze Zeit, um dann zusammen mit der Nachricht READY wieder zu erscheinen, noch ehe die Formatierung abgeschlossen ist; Sie können dann schon während der Formatierung mit dem Bildschirm arbeiten.

Geben Sie nach Abschluß der Formatierung folgenden Befehl ein:

```
LOAD "$0",8
```

Mit $0 wird das Inhaltsverzeichnis der Diskette in Laufwerk 0 angesprochen, entsprechend gilt $1 für die Diskette in Laufwerk 1. Jedes Inhaltsverzeichnis beginnt mit einem Kopfeintrag, der den Namen der Diskette und ihre Kennzahl wiedergibt. Wenn das Inhaltsverzeichnis in den Computer geladen ist, bringen Sie es mit:

```
LIST
```

auf dem Bildschirm zur Darstellung. Da wir nur eine neue Diskette formatiert haben, besteht das Inhaltsverzeichnis nur aus dem Kopfeintrag:

```
LIST

0 "YAK                 " 01 2A
664 BLOCKS FREE.

READY.
```

Auf eine formatierte Diskette können Sie jetzt Programme und Daten speichern.

PROGRAMMSPEICHERUNG: SAVE

BASIC < 3.0

Mit der Anweisung SAVE können Sie ein Programm auf einer Diskette speispeichern. SAVE wird in folgender Form angewandt:

SAVE" lfw: Programmname**", 8**

mit:

lfw	Laufwerknummer 0 oder 1
Programmname	Name, unter dem das Programm gespeichert werden soll

Die Angabe der Laufwerknummer 0 oder 1 vor dem Programmnamen ist unbedingt erforderlich. Nach dem Programmnamen folgt die Geräteadresse eines Floppy-Disk-Speichers: Fehlt die Zahl 8, dann versucht der Computer, das Programm im Bandgerät # 1 zu speichern.

Wir machen den Versuch, das Programm BLANKET auf einer Diskette in Laufwerk 0 abzuspeichern. Nach Formatierung einer Diskette und Initiierung des Laufwerks 0 geben wir das Programm BLANKET über das Tastenfeld ein. Mit der Anweisung SAVE (und nachfolgender RETURN-Taste) speichern wir dieses Programm auf der Diskette ab:

```
OPEN 1,8,15 <-------- Öffnen des Datenkanals
PRINT#1,"I0" <------- Initiieren von Laufwerk 0
                      im Floppy Disk-Gerät
10 FOR I=1 TO 800
20 PRINT "A"          Programmeingabe
30 NEXT I
40 PRINT "PHEW!"
50 END

SAVE "0:BLANKET",8 <-- Programmspeicherung
                       auf Diskette in Laufwerk 0
READY.
```

Die Speicherung des Programms auf der Diskette ist abgeschlossen, wenn der Kursor wieder auf dem Bildschirm erscheint und die Nachricht READY ausgegeben wird. Danach sollte die fehlerfreie Speicherung des Programms geprüft werden.

PRÜFUNG EINER PROGRAMMSPEICHERUNG: VERIFY — BASIC < 3.0

Die Prüfung der fehlerfreien Speicherung eines Programms auf Diskette verläuft wie bei Kassetten. Der Computer vergleicht das Programm in seinem Speicher mit dem Programm auf der Diskette und meldet Abweichungen.

Die Prüfung auf fehlerfreie Speicherung mit der Anweisung VERIFY sollte immer unmittelbar nach einer Programmspeicherung mit SAVE folgen. VERIFY wird in folgender Form angewandt:

VERIFY" lfw: Programmname", 8

mit:

lfw	Floppy Disk-Laufwerk 0 oder 1
Programmname	Name des gerade gespeicherten Programms

Speicherung des Programms BLANKET auf eine Diskette in Laufwerk 0 und anschließende Prüfung der Speicherung werden von folgenden Eingaben geleistet:

```
SAVE "0:BLANKET",8 <-------- Programm BLANKET auf
                             Diskette speichern
READY.

VERIFY "0:BLANKET",8 <------ erfolgreiche Speicherung
                             prüfen
SEARCHING FOR 0:BLANKET
VERIFYING <----------------- Speicherung erfolgreich
OK

READY.
```

Liefert die Prüfung der Speicherung eine Fehlermeldung, dann wiederholen Sie die Prüfung VERIFY. Verschwindet die Fehlermeldung nicht, dann wiederholen Sie die Speicherung und anschließende Prüfung.

Für ein gerade gespeichertes Programm kann VERIY auch in kürzerer Form eingegeben werden:

VERIFY"•",8

Das Sternsymbol (*) gibt dem CBM-Computer an, die Speicherung eines eben abgespeicherten Programms ohne Angabe des Programmnamens zu überprüfen.

LADEN EINES PROGRAMMS VON DISKETTE IN BASIC 4.0

Für das Laden eines Programms von einer Diskette in den Speicher des Computers sind in BASIC 4.0 zwei Schritte erforderlich:

1. Darstellen des Inhaltsverzeichnisses der Diskette.
2. Laden des Programms.

Zum Erproben dieser Schritte können Sie irgendeine beschriebene Diskette verwenden. Wir benutzen in unseren Beispielen Demonstrations-Disketten, die den CBM Floppy-Disk-Speichern beigefügt sind.

In BASIC 4.0 werden Laufwerke automatisch initiiert, ehe eine Programmspeicherung stattfindet. Wenn Sie jedoch die Diskette in einem Laufwerk gegen eine Diskette mit der gleichen Kennzahl (das ist die Zahl nach dem Diskettennamen im Kopfeintrag des Inhaltsverzeichnisses) austauschen, dann sollten Sie dieses Laufwerk vom Tastenfeld aus erneut initiieren, da sonst der Computer den Austausch der Disketten übersieht.

DARSTELLEN DES INHALTSVERZEICHNISSES: DIRECTORY — BASIC 4.0

Mit der Anweisung DIRECTORY wird das Inhaltsverzeichnis einer Diskette in den Speicher des Computers geladen und auf dem Bildschirm zur Darstellung gebracht. Die Anweisung wird in folgender Form eingegeben:

DIRECTORY D0	**Inhaltsverzeichnis darstellen der Diskette in**	**Laufwerk 0**
DIRECTORY D1		**Laufwerk 1**
DIRECTORY		**Laufwerke 0 und 1**

Bei fehlender Laufwerknummer werden die Inhaltsverzeichnisse beider Disketten geladen und ausgegeben. Die Anweisung DIRECTORY kann nicht in einem normalen CBM BASIC-Programm ausgeführt werden.

Hier das Beispiel eines ausgegebenen Inhaltsverzeichnisses:

```
DIRECTORY D0

 0 "8050 DEMO          " V3 2A
 5    "UNIVERSAL WEDGE"  PRG
 8    "UNIT TO UNIT"     PRG
 3    "CHANGE 8050"      PRG
 11   "COPY 2040 - 8050" PRG
 27   "PRINTER DEMO"     PRG
 12   "SEQUENTIAL"       PRG
 11   "PERFORMANCE TEST" PRG
 5    "CHECK DISK"       PRG
 17   "LOGIC DIAGNOSTIC" PRG
 1953 BLOCKS FREE.
READY.
```

LADEN EINES PROGRAMMS: DLOAD — BASIC 4.0

Mit der Anweisung DLOAD können Programme von einer Diskette in den Speicher des Computers geladen werden. Geben Sie DLOAD in direktem Dialog in einer der folgenden Formen ein:

DLOAD "Programmname"
DLOAD "Programmname",D0 Programm aus Laufwerk 0 laden

DLOAD "Programmname",D1 Programm aus Laufwerk 1 laden

Der in Anführungszeichen stehende Programmname muß unbedingt angegeben werden. Bei fehlender Laufwerknummer, D0 oder D1, wird das Programm automatisch in Laufwerk 0 gesucht. Wird ein Programm nicht gefunden, dann erscheint die Fehlermeldung ? FILE NOT FOUND.

Machen Sie einen Versuch mit der CBM DEMO-Diskette in Laufwerk 0:

```
DIRECTORY D0
```

Der Kursor sollte für kurze Zeit verschwinden, während das Inhaltsverzeichnis in den Speicher des Computers geladen und dargestellt wird.

Als nächstes laden wir das zweite Programm im Inhaltsverzeichnis (siehe Ausdruck des Inhaltsverzeichnisses im vorigen Abschnitt):

```
DLOAD "UNIT TO UNIT"
```

Danach erscheinen die gewohnten Such- und Lademeldungen:

```
DLOAD "UNIT TO UNIT",D0

SEARCHING FOR UNIT TO UNIT
LOADING
READY.
```

PROGRAMMAUFLISTUNG: LIST — BASIC 4.0

Mit der Anweisung LIST wird ein von der Diskette geladenes Programm auf dem Bildschirm dargestellt. Form und Anwendung von LIST wurde bereits früher ausführlich beschrieben.

PROGRAMMABLAUF: RUN — BASIC 4.0

Durch Eingabe der Anweisung RUN und nachfolgendem Drücken der Taste RETURN wird das eben geladene Programm zum Ablauf gebracht:

```
RUN
```

LADEN UND ABLAUFEN EINES PROGRAMMS — BASIC 4.0

Durch Bedienen der beiden Tasten SHIFT und RUN/STOP kann das erste Programm auf der Diskette in Laufwerk 0 in den Computer geladen und unmittelbar danach zum Ablauf gebracht werden.

Die Kombination von Laden und Start eines Programms auf Diskette ist nur auf CBM-Computern mit BASIC 4.0 möglich.

Hier das Aussehen der Eingaben und die Rückmeldungen des Computers:

```
DL"*          <-- Tasten SHIFT und RUN gedrückt

SEARCHING FOR 0:*
LOADING
              <-- Programmablauf beginnt
```

Laden und Start eines Programms über die Tasten SHIFT und RUN kann nur in direktem Dialog mit einem CBM-Computer ausgeführt werden. Werden diese Tasten während eines Programmlaufs bedient, dann wirken sie wie ein STOP und unterbrechen den Programmlauf.

ANLEGEN EINER LEEREN DISKETTE: HEADER — BASIC 4.0

Die Oberfläche einer fabrikneuen Diskette muß für die Speicherung von Daten vorbereitet oder "formatiert" werden, ehe die Diskette verwendet werden kann. Mit den folgenden Eingaben wird eine neue Diskette formatiert bzw. die Formatierung einer alten Diskette wiederholt:

HEADER "Diskname", **Dx,Iyy**

wobei:

Diskname	Benennung (Name) der Diskette
x	Floppy Disk-Laufwerk 0 oder 1
yy	2stellige beliebige Kennzahl

Ein Beispiel für die Anwendung von HEADER: Wir geben der Diskette den Namen YAK und die Kennzahl 01. Die Diskette befindet sich in Laufwerk 0 unseres Floppy-Disk-Speichers. Dann wird diese Diskette mit folgender Anweisung angelegt:

```
HEADER "YAK", D0, I01
```

Wenn Sie jetzt die Taste RETURN drücken, antwortet der Computer mit der Frage: Sind Sie sicher? (ARE YOU SURE?). Wenn Sie sicher sind, geben Sie "Y" für YES ein und drücken die Taste RETURN. Mit "N" für NO und der Tastenangabe RETURN löschen Sie Ihre HEADER-Eingabe.

Nach Eingabe von "Y" leuchtet die Anzeige am Laufwerk 0 auf, während die Diskette formatiert wird. Am Ende der Formatierung erscheinen die Nachricht READY und der Kursor:

```
HEADER "YAK", D0, I01
ARE YOU SURE ?Y ←——— Eingabe von y (für YES)
READY.                und RETURN
▒
```

Nach der Formatierung können Sie beginnen, Programme und Daten auf der Diskette zu speichern.

PROGRAMMSPEICHERUNG: DSAVE — BASIC 4.0

Mit der Anweisung DSAVE können Sie ein Programm im Speicher Ihres Computers auf einer Diskette abspeichern. Hierfür geben Sie in direktem Dialog die Anweisung in folgender Form ein:

DSAVE "Programmname",Dx

mit:

Programmname	Name, unter dem das Programm gespeichert werden soll
x	Floppy Disk-Laufwerke 0 oder 1

Der Programmname in Anführungszeichen ist unbedingt erforderlich. Bei fehlender Laufwerknummer wird das Programm automatisch auf der Diskette in Laufwerk 0 abgespeichert.

Machen Sie einen Versuch mit dem Programm BLANKET. Legen Sie eine formatierte Diskette in Laufwerk 0 ein. Geben Sie das Programm BLANKET ein und veranlassen Sie eine Speicherung auf der Diskette mit folgender Anweisung:

```
DSAVE "BLANKET",D0
```

Nach Drücken der Taste RETURN leuchtet die Anzeigelampe am Laufwerk 0 auf und der Kursor verschwindet vom Bildschirm. Am Ende der Speicherung erscheinen die Nachricht READY und der Kursor:

```
DSAVE "BLANKET",D0

READY.
▒
```

Danach sollte die Prüfung auf fehlerfreie Speicherung erfolgen.

PRÜFUNG AUF FEHLERFREIE SPEICHERUNG: VERIFY — BASIC 4.0

Die Anweisung VERIFY zur Prüfung einer eben erfolgten Programmspeicherung hat in BASIC 4.0 die gleiche Form wie in BASIC < 3.0. Entnehmen Sie die Beschreibung VERIFY den Angaben unter BASIC < 3.0.

ARBEITEN MIT DEM TEXTDRUCKER

CBM-Drucker eignen sich hervorragend zur Textausgabe, zur Darstellung von Ergebnissen und zum Ausdruck von Programmen. Das Arbeiten mit Druckern erfolgt wie bei Floppy-Disk-Speichern und Bandgeräten über Anweisungen, die man entweder am Tastenfeld eingibt, oder im Zusammenhang mit einem Programm verwendet. Wir werden hier das Arbeiten mit Druckern in direktem Dialog mit dem Computer beschreiben, während programmgesteuerte Druckerausgaben ausführlich in Kapitel 6 beschrieben sind.

Zwischen BASIC 4.0 und BASIC < 3.0 gibt es keine Unterschiede in den Anweisungen zum Arbeiten mit CBM-Druckern.

ÖFFNEN EINES DATENKANALS: OPEN

Mit der Anweisung OPEN wird zunächst ein Datenkanal vom Computer zum Drucker geöffnet, ehe Daten zum Drucker gesendet werden können:

OPEN x,4

mit:

x	Ganzzahl aus dem Bereich 0 bis 255

Hier Beispiele für OPEN-Anweisungen aus der Praxis:

```
OPEN 1,4

OPEN 4,4

OPEN 250,4
```

OFFENHALTEN DES DATENKANALS: CMD

Mit der Anweisung CMD wird ein von OPEN geöffneter Datenkanal für den nachfolgenden Datenverkehr offengehalten. Eine ausführliche Erklärung der Wirkungsweise CMD finden Sie in Kapitel 6. Die Anweisung hat folgende Form:

```
CMD 1
```

Die Zahl nach CMD muß mit der ersten Zahl nach OPEN übereinstimmen. Ist keine Übereinstimmung gegeben, dann erscheint die Fehlermeldung ? FILE NOT OPEN ERROR, da in diesem Fall mit CMD ein Datenkanal offengehalten werden sollte, der mit OPEN nicht geöffnet wurde. Nach dieser Fehlermeldung geben Sie die Anweisungen OPEN und CMD erneut ein.

Hier finden Sie Beispiele, wie man die Datenverbindung zum Drucker benutzt, um Daten in Form von "Zeichenketten" ausdrucken zu lassen:

```
OPEN 1,4:CMD 4
PRINT "SPACE SHUTTLE"

OPEN 2,4
CMD 2, "MEIN CBM BEISST"

OPEN 3,4
PRINT#3, "1234567890"
```

DATENÜBERTRAGUNG ZUM DRUCKER: PRINT

Für die Übertragung von Daten zum Drucker verwenden wir die Anweisung PRINT in direktem Dialog mit dem Computer.

Vergewissern Sie sich zunächst, daß Drucker und Computer miteinander verbunden sind, das Farbband eingelegt und das Druckerpapier vorschriftsmäßig eingeführt sind. Danach öffnen Sie die Datenverbindung zum Drucker mit folgender Angabe:

```
OPEN 1,4:CMD 1
```

Nach Eingabe der Taste RETURN führt der Drucker einen Zeilenvorschub aus. Jetzt ist der Drucker bereit, am Tastenfeld eingegebene Daten auszudrucken. Lassen Sie ihn Ihren Namen ausdrucken:

```
PRINT "KIT CARSON"
```

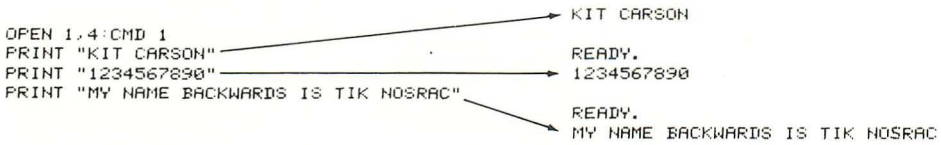
Wenn Sie die RETURN-Taste drücken, verschwindet der Kursor vom Bildschirm, während Ihr Name etwa so ausgedruckt wird:

```
READY.
KIT CARSON
```

Beim Arbeiten mit dem Drucker in direktem Dialog ist die erste ausgedruckte Nachricht READY. Auf der nächsten Zeile erscheint dann Ihre Ausgabe, im Beispiel Ihr Name. Danach bewegt sich der Druckkopf auf den folgenden Zeilenanfang.

Hier folgen Beispiele für Eingabe und Druckerausgabe:

Eingabe (Bildschirm): **Druckerausgabe:**



PROGRAMMAUSDRUCK: LIST

Mit der Anweisung LIST kann ein Programm aus dem Speicher des Computers auf dem Drucker ausgegeben werden. Durch vorangehende OPEN- und CMD-Anweisungen wird die Auflistung des Programms vom Bildschirm zum Drucker umgelenkt:

```
OPEN 1,4:CMD 1
LIST
```

Nach LIST können, wie gewohnt, Zeilennummern angegeben werden. Das folgende Beispiel zeigt den Ausdruck des Programms BLANKET:

```
OPEN 1,4:CMD 1
LIST

 10 FOR I=1 TO 800
 20 PRINT "A";
 30 NEXT I
 40 PRINT "PHEW!"
 50 END
```

SCHLIESSEN DES DATENKANALS: CLOSE

Mit der Anweisung CLOSE müssen Sie jeden beendeten Datenverkehr mit dem Drucker abschließen:

```
CLOSE 1
```

Die Zahl nach CLOSE muß übereinstimmen mit der Zahl nach OPEN:

```
OPEN 1, 4                OPEN 15, 4
.                        .
.                        .
.                        .
CLOSE 1                  CLOSE 15
```

Die Anweisung CLOSE wird erst nach einer PRINT#-Anweisung wirksam. Die folgenden Beispiele zeigen richtige und falsche Anwendungen der Anweisung CLOSE:

Richtig **Falsch**

```
OPEN 5, 4
PRINT#5,"HELLO THERE"
CLOSE 5
```

Richtig	Falsch	
OPEN 5, 4 CMD 5,"HELLO THERE" PRINT#5:CLOSE 5	OPEN 5, 4 CMD 5, "HELLO THERE" CLOSE 5	es fehlt: ← PRINT #5
OPEN 5,4 CMD 5,"HELLO THERE" PRINT#5,"HELLO THERE" CLOSE 5	OPEN 5,4 CMD 5,"HELLO THERE" PRINT#5,"HELLO THERE" PRINT#5:CLOSE 5	überflüssig: ← PRINT #5
OPEN 5, 4 PRINT#5,"HELLO THERE" CMD5,"HELLO THERE" PRINT#5:CLOSE 5	OPEN 5, 4 PRINT#5,"HELLO THERE" CMD5,"HELLO THERE" CLOSE 5	es fehlt: ← PRINT #5

KAPITEL 3

Textbearbeitung am Bildschirm

CBM Computer stellen jedes am Tastenfeld eingegebene Zeichen sofort auf dem Bildschirm dar. Jede Bildschirmdarstellung kann in direktem Dialog verändert, d.h. einer Edition unterzogen werden.

Bildschirmedition ist eine der bedeutendsten Fähigkeiten Ihres CBM Computers. Sie können Ihre Texteingabe einfach und schnell mit Hilfe des eingebauten Editionsprogramms ändern, von dessen Beschreibung dieses Kapitel handelt. Erproben Sie alle in diesem Kapitel beschriebenen Beispiele. Sollten Sie diese Editionsprozeduren nicht verstehen, dann lesen Sie zunächst Kapitel 4, "Programmierung der CBM Computer". Kapitel 4 führt in die Grundlagen der hier verwendeten Programmiersprache BASIC ein.

Das Editionsprogramm ermöglicht die freie Bewegung des Kursors über den Bildschirm in alle vier Richtungen. Zeichen können an jeder Stelle des Bildschirms gelöscht oder eingesetzt werden. Die Kursortasten wurden in Kapitel 1 unter "Funktion der Eingabetasten" beschrieben: Die Taste CLEAR SCREEN/HOME führt den Kursor auf die oberste Zeile des Bildschirms zurück und/oder löscht alle Bildschirmdarstellungen. Die Tasten CURSOR UP/DOWN und CURSOR LEFT/RIGHT verschieben den Kursor über die Textdarstellung. Die Taste INSERT/DELETE schafft Platz für Einfügungen einzelner Zeichen oder löscht Zeichen.

DIREKTE TEXTKORREKTUR AM BILDSCHIRM

Häufig werden Sie Fehler schon während des Schreibens einer Zeile bemerken. Sie können solche Fehler sofort korrigieren. Bewegen Sie den Kursor zum fehlerhaften Zeichen zurück, indem Sie die Taste CURSOR LEFT oder DELETE benutzen.

Zur Illustration folgender Text:

```
MEIN CBM BAISST
```

BEISST ist verschrieben; wir wollen das A in ein E ändern.

Sie können die Taste CURSOR LEFT benutzen, um den Kursor auf das A zu bewegen, ohne daß dabei der überfahrene Text verändert wird. Oder Sie benutzen die Taste DELETE für die Rückführung des Kursors, wobei der Text bis hin zum A gelöscht wird. Die Entscheidung hängt davon ab, ob Sie den Text rechts vom A be-

wahren wollen. Liegt der Eingabefehler viele Zeichen zurück, dann ist die Taste CURSOR LEFT ratsam, so daß die Zeile nach dem Eingabebefehl nicht erneut eingegeben werden muß, möglicherweise mit neuen Fehlern.

KURSORRÜCKFÜHRUNG MIT DER TASTE "CURSOR LEFT"

Geben Sie folgenden Text ein, ohne die RETURN Taste zu drücken, d.h. der Kursor bleibt am Textende stehen:

```
MEIN CBM BAISST▒
```

Der Kursor muß zum A zurückgeführt werden, um das A in ein E zu ändern. Um den Kursor nach links zu bewegen, drücken Sie gleichzeitig die Tasten CURSOR LEFT und SHIFT. Je Tastenbedienung bewegt sich der Kursor um eine Position nach links, bis Sie ihn über dem A zum Halten bringen.

Auf CBM 8000 Modellen bewegt sich der Kursor automatisch nach links, solange Sie die beiden Tasten gedrückt halten.

```
MEIN CBM BAISST▒
MEIN CBM BAISS▒
MEIN CBM BAIS▒T
MEIN CBM BAI▒ST
MEIN CBM BA▒SST
MEIN CBM B▒ISST
```

Ändern Sie jetzt das A in ein E durch Druck auf die Taste E. Danach bewegt sich der Kursor um eine Stelle nach rechts. Diese Korrekturart wird "Überschreiben" genannt, da der Kursor das vorhandene Zeichen mit dem neuen überschreibt. Überschreiben ändert den Text am Bildschirm zu:

```
MEIN CBM BE▒SST ←
```
Kursor nach überschreiben des A mit einem E

Sie können jetzt den Kursor durch mehrfaches Drücken der Taste CURSOR RIGHT an das Ende des Textes bewegen, um die Eingabe fortzusetzen. Oder Sie drücken einfach die Taste RETURN. Erproben Sie beide Möglichkeiten.

Sobald die Edition einer Zeile abgeschlossen ist, werden durch Druck der Taste RETURN die Änderungen endgültig. Bewegen Sie den Kursor nicht von der Zeile fort, etwa durch Drücken der Taste HOME, CLEAR SCREEN oder CURSOR UP/DOWN. Obwohl der Bildschirm eine durchgeführte Textänderung zeigt, wird die Änderung erst dann im Speicher des Computers registriert, wenn die Taste RETURN gedrückt worden ist.

KURSORRÜCKFÜHRUNG MIT DER TASTE "DELETE"

Geben Sie den Beispieltext noch einmal ein, wobei Sie wieder den Kursor am Textende stehen lassen. Drücken Sie nicht die RETURN Taste.

```
MEIN CBM BAISST▒
```

Drücken Sie jetzt die DELETE Taste so oft, bis der Kursor das A gelöscht hat:

```
MEIN CBM BAISST▒ ←
MEIN CBM BAISS▒ ←
MEIN CBM BAIS▒ ←  wiederholtes Drücken der Taste DELETE
MEIN CBM BAI▒ ←
MEIN CBM BA▒ ←
MEIN CBM B▒ ←
```

Der Kursor sollte über dem A zum Halt kommen. Drücken Sie jetzt die E Taste, um das E einzugeben.

Um den Satz zu beenden, müssen Sie den gelöschten Text erneut eingeben. Nach Beendigung drücken Sie die RETURN Taste, so daß die Änderung im Computer abgespeichert wird.

```
MEIN CBM B▒          Eingabe der Buchstaben
MEIN CBM BE▒ ←       I
MEIN CBM BEI▒ ←      S
MEIN CBM BEIS▒ ←     S
MEIN CBM BEISS▒ ←    T
MEIN CBM BEISST▒ ←   Korrektur beendet
```

TEXTLÖSCHUNG MIT DER TASTE "DELETE"

Die DELETE Taste kann auch dazu benutzt werden, einen Text um eine Stelle nach links zu verschieben und dabei das links stehende Zeichen zu löschen.

Im Textbeispiel wurde ein zusätzlicher Buchstabe eingefügt.

```
MEIN CBM BEIISST▒
```

Geben Sie diesen Satz ein, mit dem Kursor am Textende. Unsere Aufgabe ist, das zusätzliche I in BE(I)ISST zu löschen. Anstatt den Text bis zum I hin zu löschen, schieben Sie den Kursor mit der CURSOR LEFT Taste über den Text bis zum zweiten I. Mit DELETE wird das Zeichen unmittelbar links vom Kursor gelöscht, d.h. (I), nicht jedoch das Zeichen unter dem Kursor.

```
MEIN CBM BEI▒SST
```

Drücken Sie die DELETE Taste, wenn der Kursor auf dem zweiten I liegt: Das erste (I) wird gelöscht und der Text um eine Stelle nach links verschoben, womit der Zwischenraum wieder gefüllt ist.

```
MEIN CBM BEI▒SST  ← Kursor vor Drücken der TASTE DELETE
MEIN CBM BE▒SST   ← Taste DELETE gedrückt
MEIN CBM BEISST▒  ← Kursor nach rechts verschoben
```

Drücken Sie jetzt die CURSOR RIGHT Taste mehrere Male oder einfach die RETURN Taste, um den Text wieder zu verlassen.

TEXTEINFÜGUNG MIT DER TASTE "INSERT"

Die INSERT Taste schafft an der augenblicklichen Stellung des Kursors Platz, indem sie den Text um eine Stelle nach rechts schiebt und die Eingabe eines neuen Zeichens ermöglicht.

Zur Demonstration der INSERT Taste werden wir in unserem Textbeispiel ein Zeichen auslassen. Geben Sie nachfolgenden Satz ein, wieder mit dem Kursor am Textende:

```
MEIN CBM BIST
```

Das fehlende E kann auf verschiedene Weisen eingefügt werden.

Sie können den Text vom Ende her bis zur Fehlerstelle löschen und die Eingabe richtig wiederholen, wie zuvor beschrieben wurde. Wenn Sie aber viele Zeichen zu löschen haben, könnten Sie bei der erneuten Eingabe andere Fehler machen. Das entfällt mit der INSERT Taste.

Bevor Sie die INSERT Taste gebrauchen, bewegen Sie den Kursor bis zu der Stelle, an der Sie die Einfügung beginnen lassen wollen. Im Textbeispiel steht der Kursor dann auf dem ersten Zeichen des unverändert bleibenden Textteils, der, um Platz zu schaffen, nach rechts wandert:

```
MEIN CBM BST  <-- Kursor vor Drücken der Taste INSERT
                  nach Drücken von INSERT hier ein E eingeben
```

Zum Einfügen des E zwischen B und I bewegen Sie den Kursor mit der CURSOR LEFT Taste zurück, bis er auf dem I steht:

```
MEIN CBM BISST
MEIN CBM BISS  <--
MEIN CBM BIS T <--
MEIN CBM BI ST <--   wiederholtes Drücken der Taste CURSOR LEFT
MEIN CBM B SST <--
```

Drücken Sie die INSERT Taste einmal, entsprechend dem Platz für ein Zeichen. Der Textteil ISST bewegt sich um eine Stelle nach rechts, während der Kursor stehen bleibt.

```
MEIN CBM B ISST  <-- INSERT Taste verschiebtden Wortteil ISST nach rechts
```

Geben Sie nun das Zeichen E ein:

```
MEIN CBM BE SST  <-- Buchstabe E eingeben
```

Wenn noch weiterer Text auf dieser Zeile folgen soll, bewegen Sie den Kursor mit CURSOR RIGHT an das Textende, geben weiteren Text ein, und drücken dann die RETURN Taste. Oder Sie drücken gleich die RETURN Taste, wenn kein weiterer Text auf dieser Zeile folgt und der Kursor auf den Zeilenanfang der folgenden Zeile springen soll.

Im folgenden fügen wir ein ganzes Wort ein. Geben Sie den nachfolgenden Text ein, mit dem Kursor am Textende:

```
JETZT IST ES ZEIT
```

Nehmen Sie an, wir wollen das Wort NICHT einfügen, um den Sinn des Satzes zu ändern:

```
JETZT IST ES ZEIT
             ↑
           nicht
```

Drücken Sie die Taste CURSOR LEFT wiederholt, bis der Kursor auf dem Z des Wortes Zeit zu stehen kommt:

```
JETZT IST ES ZEIT
JETZT IST ES ZEI
JETZT IST ES ZE T
JETZT IST ES Z IT
JETZT IST ES  EIT
```

Drücken Sie nun die INSERT Taste sechs mal, um für das Wort NICHT sowie einen Zwischenraum Platz zu schaffen:

```
JETZT IST ES  EIT
JETZT IST ES  ZEIT ←
JETZT IST ES   ZEIT ←
JETZT IST ES    ZEIT ←  INSERT Taste wiederholt gedrückt
JETZT IST ES     ZEIT ←
JETZT IST ES      ZEIT ←
JETZT IST ES       ZEIT ←
```

Geben Sie das Wort NICHT und einen Zwischenraum ein:

```
JETZT IST ES        ZEIT
JETZT IST ES N      ZEIT
JETZT IST ES NI     ZEIT
JETZT IST ES NIC    ZEIT
JETZT IST ES NICH   ZEIT
JETZT IST ES NICHT  ZEIT
```

Mit RETURN können Sie jetzt den Kursor aus dem Text entfernen.

Es gibt einige Regeln zur Benutzung der INSERT Taste. Bewegen Sie immer den Kursor bis zu der Stelle, an der Sie die Einfügung beginnen lassen wollen. Das Zeichen unter dem Kursor wird dann nach rechts verschoben und ist später das erste Zeichen hinter der Einfügung. Geben Sie so viele Zeichen und Zwischenräume ein, wie Sie INSERT Tastenbedienungen eingegeben hatten.

TEXTKORREKTUR ZWISCHEN "ANFÜHRUNGSZEICHEN"

Die Edition von Texten zwischen Anführungszeichen erfordert eine abweichende Prozedur, da Anführungszeichen Anfang und Ende einer Zeichenkette signalisieren.

Erinnern Sie sich aus Kapitel 2, daß jeder Text, der in einer ungeraden Anzahl von Anführungszeichen eingeschlossen ist, eine Zeichenkette bedeutet. Wenn Sie versuchen, den Kursor zur Textedition zu benutzen, werden Sie feststellen, daß er sich nicht bewegen läßt. Stattdessen wird der Kursor durch ein Symbol dargestellt, das Teil der Zeichenkette wird. Bevor die Edition einer Zeichenkette beginnen kann, muß sie durch ein zweites Anführungszeichen abgeschlossen sein, oder die Taste RETURN gedrückt worden sein. Sobald die Zeichenkette verlassen wurde, arbeitet der Kursor wieder normal. Den Modellen CBM 8000 unterdrückt die ESC Taste die Wirkung einer ungeraden Anzahl von Anführungszeichen.

Die Edition von Zeichenketten wird an folgendem Text demonstriert:

```
PRINT"MEIN CBM BEISST"
```

Dies ist eine PRINT Anweisung, die die Zeichenkette "mein CBM beisst" auf der nächsten Zeile des Bildschirms darstellt, sobald die RETURN Taste bedient wurde.

Nach Abschluß der PRINT Anweisung mögen wir feststellen, daß das E in BEISST versehentlich als A eingegeben wurde:

```
PRINT"MEIN CBM BAISST"
```

Sie sind vielleicht versucht, die CURSOR LEFT Taste zu benutzen, um das A mit einem E zu überschreiben. Das wird nicht gehen. Die CURSOR LEFT Eingaben werden als Symbole in der Zeichenkette dargestellt, statt den Kursor zu bewegen. Erst bei Ausführung der PRINT Anweisung kommt es zu einer Kursorbewegung:

```
PRINT"MEIN CBM BEISS
```

CURSOR LEFT Taste zwei mal gedrückt

Tabelle 3-1 zeigt, mit welchen Symbolen die Kursortasten in einer Zeichenkette dargestellt werden. Um ein derartiges "Programmieren" der Kursortasten in einer Zeichenkette zu vermeiden, geben Sie ein zweites Anführungszeichen ein, bevor Sie mit der Textedition beginnen.

Angewandt auf unser Beispiel: beenden Sie zunächst die Eingabe der Zeichenkette und schließen Sie mit Anführungszeichen ab:

```
PRINT"MEIN CBM BAISS
PRINT"MEIN CBM BAISST
PRINT"MEIN CBM BAISST"
```

Kursorstellung nach Eingabe demZweiten Anführungszeichen

Tabelle 3 - 1.Symbole programmierter Tastenfunktionen

Funktion	Tasteneingabe	Symbol
DELETE	INST DEL	nicht programmierbar
INSERT	SHIFT + INST DEL	nicht ausführbar
HOME	CLR SCREEN HOME	
CLEAR	SHIFT + CLR SCREEN HOME	
KURSOR ↓	CURSOR	
KURSOR ↑	SHIFT + CURSOR	
KURSOR →	CURSOR	
KURSOR ←	SHIFT + CURSOR	

Jetzt erst bewegen Sie den Kursor mit CURSOR LEFT zurück zum A in BAISST:

```
PRINT"MEIN CBM BAISST"
PRINT"MEIN CBM BEISST"  ←
PRINT"MEIN CBM BAISS"   ←
PRINT"MEIN CBM BAIST"   ← wiederholtes Drücken der Taste
PRINT"MEIN CBM BAIST"   ← CURSOR LEFT
PRINT"MEIN CBM BASST"   ←
PRINT"MEIN CBM BISST"   ←
```

Überschreiben Sie das A mit einem E und verlassen Sie den Satz durch Drücken der Taste RETURN. Die korrigierte Zeichenkette wird dann auf der folgenden Bildschirmzeile dargestellt:

```
PRINT "MEIN CBM BEISST"  ← A durch E überschreiben
PRINT "MEIN CBM BESST"   ← RETURN drücken
MEIN CBM BEISST          ← korrigierte PRINT - Darstellung

READY
```

Eine andere Möglichkeit der Edition von Text zwischen Anführungszeichen ist, die Taste RETURN zu drücken, wodurch der Kursor von der augenblicklichen Zeile verschwindet.

Hierdurch wird der Kursor auf die erste Zeichenposition der folgenden Zeile gesetzt. (Der Computer mag auf verschiedene Weise antworten, abhängig von der Anweisung. Machen Sie sich hierüber keine Gedanken – Ihr Interesse gilt jetzt der Textedition.) Bewegen Sie den Kursor nach oben mit CURSOR UP, und dann nach rechts zur Stelle der Edition.
Geben Sie folgende Anweisung ein:

```
PRINT"MEIN CBM BAISST"
```

Durch Drücken der RETURN Taste springt der Kursor auf die nächste Zeile.

```
PRINT"MEIN CBM BAISST"
MEIN CBM BAISST                ← nach Drücken der Taste RETURN

READY
▒
```

Ohne Beachtung der Bildschirmantwort bewegen Sie jetzt den Kursor nach oben zur ursprünglichen Anweisungszeile, indem Sie CURSOR UP wiederholt betätigen:

wiederholtes Drücken der Taste CURSOR UP

```
▒RINT"MEIN CBM BAISST"
MEIN CBM BAISST

READY
▒
```

Drücken Sie die CURSOR RIGHT Taste, um mit dem Kursor zum Zeichen A in BAISST zu gelangen. Überschreiben Sie A durch E.

```
PRINT"MEIN CBM B▒ISST"
PRINT"MEIN CBM BE▒SST"   ← A überschrieben mit einem E
```

Nach Drücken der RETURN Taste wird die Anweisung ausgeführt. Die Zeichenkette wird mit dem neuen Zeichen E statt A auf dem Bildschirm dargestellt:

```
PRINT"MEIN CBM BE▒SST"   ← Kursor nach Korrektur des A
MEIN CBM BEISST          ← nach Drücken der RETURN Taste

READY.
▒
```

Wie die CURSOR LEFT Taste werden auch die Tasten CURSOR RIGHT und CURSOR UP/DOWN als Symbole (siehe Tabelle 3-1) in einer Zeichenkette dargestellt, wenn sie nach einer ungeraden Zahl von Anführungszeichen eingegeben wurden.

Lediglich die Tasten INSERT/DELETE lösen eine andere Reaktion aus.

Die INSERT Taste ergibt ein Symbol in einer Zeichenkette, wenn die Eingabe nach einer ungeraden Anzahl von Anführungszeichen erfolgte. Auf dem Bildschirm wird sie durch das INSERT Symbol ("▌") dargestellt. Wird dann aber der PRINT Befehl ausgeführt, hat das INSERT Symbol keine Auswirkung auf die Bildschirmdarstellung.

Die DELETE Taste bleibt als einzige unbeeinflußt von der Wirkung der Anführungszeichen. Zur Edition unseres Textbeispiels mit der DELETE Taste brauchen Sie lediglich die Taste drücken, um den Text bis zu der Stelle zu löschen, an der die Edition stattfinden soll:

```
PRINT"MEIN CBM BAISST"▒
PRINT"MEIN CBM BAISST▒   ←
PRINT"MEIN CBM BAISS▒    ←
PRINT"MEIN CBM BAIS▒     ←   wiederholtes Drücken der Taste DELETE
PRINT"MEIN CBM BAI▒      ←
PRINT"MEIN CBM BA▒       ←
PRINT"MEIN CBM B▒        ←
```

EDITION VON PROGRAMMANWEISUNGEN

Lesen Sie diesen Abschnitt erst, wenn Ihnen das Programmieren eines Computers vertrauter geworden ist.

Dieser Abschnitt erklärt die Duplizierung und Edition ähnlicher BASIC Anweisungen unter Benutzung von Zeilennummern.

DUPLIZIERUNG VON PROGRAMMZEILEN

Viele Programme haben eine Anzahl ähnlicher oder identischer Anweisungen. Es ist oft nützlich, Anweisungen von einer vorhandenen Originalanweisung durch Duplizieren herzustellen, statt sie viele Male erneut einzugeben. Auf CBM Computern wird jeder Programmzeile eine eindeutige Zeilennummer zugeordnet. Durch Ändern dieser Zeilennummer kann man eine neue Anweisung herstellen, ohne das Original zu zerstören.

Geben Sie die folgende Programmanweisung ein:

```
10 PRINT"*"   ←  Zeilennummer 10 überschrieben durch 20
```

Nehmen Sie an, wir benötigen noch fünf weitere Anweisungen von der oben angegebenen Form. Wir könnten die Anweisung noch fünf Mal eingeben und dabei jeder Anweisung ihre eigene Zeilennummer geben. Oder wir können die Anweisung duplizieren, indem wir die Zeilennummer fünf Mal ändern, wie nachfolgend beschrieben.

Geben Sie die zu duplizierende Anweisung ein. (Ist die Anweisung bereits eingegeben worden, lassen Sie sie unter Angabe Ihrer Zeilennummer mit dem Befehl LIST ausgeben.) Drücken Sie die Taste CURSOR UP, bis der Kursor am Anfang der Zeilennummer steht. Zur Änderung der Zeilennummer überschreiben Sie sie mit einer neuen Nummer:

```
20▒PRINT"*"
```

Ist die neue Nummer eingegeben, drücken Sie die RETURN Taste, womit die neue Anweisung entstanden ist. Die RETURN Taste muß nach jeder Änderung einer Zeilennummer gedrückt werden. Lassen Sie jetzt das Programm auflisten, und zwei Anweisungen erscheinen auf dem Bildschirm:

```
 20 PRINT"*"
LIST

 10 PRINT"*"
 20 PRINT"*"
READY.
```

EDITION ÄHNLICHER PROGRAMMANWEISUNGEN

Edition ähnlicher Programmanweisungen folgt dem gleichen Verfahren wie die Duplizierung von Programmzeilen. Lassen Sie die zu duplizierende Anweisung unter Angabe der Zeilennummer mit LIST ausgeben. Bewegen Sie den Kursor nach oben zur Zeilennummer und überschreiben Sie diese mit der neuen Nummer. Mit den Tasten CURSOR RIGHT, INSERT oder DELETE bewegen Sie den Kursor, um die Anweisung wie benötigt zu verändern. Nach Drücken der RETURN Taste ist die neue Anweisung geschaffen und wird eingespeichert. Machen Sie sich keine Gedanken, wenn das Original der Anweisung nicht mehr auf dem Bildschirm erscheint. Es ist noch im Speicher und wird wieder angezeigt, wenn Sie das Programm mit LIST ausgeben.

Schließlich wollen wir ein kurzes Programmbeispiel schreiben, das wie folgt aussieht:

```
10 PRINT "*"
20 PRINT "  *"
30 PRINT "    *"
40 PRINT "      *"
50 PRINT "        *"
```

Da alle fünf Zeilen ähnlich sind, wollen wir die erste Anweisung vier Mal duplizieren und jedesmal verändern, um die überflüssige Eingabe fast identischer Texte zu vermeiden.

Geben Sie die Programmanweisung 10 ein und drücken Sie die RETURN Taste. Um Programmanweisung 20 zu erzeugen, bewegen Sie den Kursor zur Zeile 10 und überschreiben die 10 durch eine 20. Dann bewegen Sie den Kursor durch wiederholtes Betätigen der Taste CURSOR RIGHT nach rechts, bis er auf dem Sternsymbol (*) liegt. Durch Druck auf die INSERT Taste bewegt sich der Stern um eine Stelle nach rechts und hinterläßt eine Leerstelle unter dem Kursor. Mit der Taste RETURN wird die neue Anweisung im Computer abgespeichert. Obwohl Zeile 10 jetzt auf dem

Bildschirm verschwunden ist, bleibt diese Anweisung im Speicher des Computers erhalten, wie die Programmausgabe mit LIST zeigt:

```
 20 PRINT "  *"      ← Zeile 20 gebildet aus Zeile 10
LIST

 10 PRINT "*"
 20 PRINT "  *"
READY.
```

Anweisungen 30, 40 und 50 werden in der gleichen Weise gebildet wie Anweisung 20. Nach Eingabe aller fünf Anweisungen sollte der LIST Befehl folgendes Programm auf dem Bildschirm ausgeben:

```
50 PRINT "       *"
LIST

10 PRINT "*"
20 PRINT "  *"
30 PRINT "    *"
40 PRINT "      *"
50 PRINT "       *"
READY.
```

Wie Sie sehen ist die Duplizierung und Edition ähnlich aussehender Programmanweisungen eine wirkungsvolle Methode zur Programmerstellung.

BILDSCHIRMEDITION IN BASIC 4.0

BASIC 4.0 verfügt über einige Editionsmöglichkeiten, die nicht in früheren Ausgaben von CBM BASIC enthalten waren. Diese zusätzlichen Möglichkeiten werden im allgemeinen im Zusammenhang mit Programmen eingesetzt, d.h. nicht zur Edition in direktem Dialog mit dem Computer, und werden deshalb in Kapitel 5 beschrieben.

KAPITEL 4

Programmierung von CBM Computern

In diesem Kapitel zeigen wir Ihnen, wie Sie Ihre eigenen Programme in BASIC schreiben können. BASIC ist eine Programmiersprache. Wie jede andere Programmiersprache besteht BASIC aus Anweisungen oder Befehlen, mit denen Sie sich Ihr Programm zusammensetzen. Ein Programm formuliert Ihre Aufgabe in einer Sprache, die der Computer versteht und in der er Ihre Aufgabe lösen kann.

Man könnte sich die Programmiersprache BASIC beibringen, indem man eine BASIC-Vokabel nach der anderen lernt. Da aber die einzelnen Vokabeln nicht viel Sinn machen, werden Sie vielleicht bald aufgeben. Ein Studium der einzelnen BASIC-Anweisungen entartet schnell in das bloße Auswendiglernen einer Reihe willkürlich erscheinender Vokabeln, die Ihnen nichts über Programmierung oder gute Programmierpraxis sagen.

Eine vollständige Zusammenstellung und Beschreibung sämtlicher BASIC-Anweisungen für CBM-Computer wurde daher zu einem eigenen Kapitel gemacht (Kapitel 8). Sehen Sie für einzelne Anweisungen in Kapitel 8 nach, wenn es nötig ist: Versuchen Sie aber nicht, Kapitel 8 vor diesem Kapitel zu lesen.

DIREKTER DIALOG UND PROGRAMMABLÄUFE

Nach dem Einschalten befinden sich CBM-Computer in einer Betriebsart, in der ein direkter Dialog mit dem Computer wie bei Taschenrechnern möglich ist; BASIC-Anweisungen werden sofort nach Eingabe von RETURN ausgeführt. Hier sind Beispiele für direkte Dialoge:

```
?4.5+6.42
 10.92
```
Addition

```
READY.
?500-410
 90
```
Subtraktion

```
READY.
?π*2
 6.28318531
```
Multiplikation

```
READY.
?100/3
 33.3333333
```
Division

Ergebnisse werden unmittelbar auf der nächsten Bildschirmzeile dargestellt.

Wird jedoch bei Eingaben jeder Bildschirmzeile eine Zeilennummer vorangesetzt, dann erwarten CBM-Computer nicht einen direkten Dialog, sondern einen Programmablauf. Der Computer speichert Ihre Eingaben, führt sie aber nicht aus, ehe er durch eine RUN-Anweisung hierzu aufgefordert wird. Die Einzelheiten der Programmeingabe werden in den folgenden Abschnitten besprochen.

PROGRAMM UND ANWEISUNGEN

Jede der fünf Anweisungen aus dem oben gezeigten direkten Dialog stellen kleine Programme dar. Ein Programm ist eine genaue und vollständige Beschreibung der Aufgabe, die ein Computer ausführen soll.

Ein Programm besteht aus einer oder mehreren Anweisungen. In unserem Beispiel bestand jedes Programm aus einer Anweisung, die meisten Programme bestehen aus hunderten oder sogar tausenden von Anweisungen.

PROGRAMMABLAUF

Man sagt, ein Programm läuft ab, wenn der Computer die im Programm genannten Anweisungen ausführt.

In direktem Dialog läuft ein Programm sofort ab, wenn die Taste RETURN gedrückt wurde. Nach Eingabe aller Anweisungen eines Programms wartet der Computer mit einem Programmablauf, bis Sie die Buchstaben RUN eingegeben und danach die RETURN-Taste gedrückt haben; die RUN-Anweisung wurde bereits in Kapitel 1 beschrieben.

PROGRAMMZEILEN UND ZEILENNUMMERN

Bei Eingabe eines Programms verteilen sich die Anweisungen auf Programmzeilen, von denen jede eine eigene Zeilennummer trägt. Fehlt die Zeilennummer, dann führt der CBM-Computer Anweisungen einer Programmzeile in direktem Dialog, d.h. bei der nächsten Eingabe von RETURN, aus. Trägt die Programmzeile dagegen eine Zeilennummer, dann erfolgt bei der nächsten Eingabe von RETURN die Speicherung dieser Programmzeile, nicht aber ihre Ausführung.

Eine Programmzeile kann bis zu 80 Zeichen lang sein. Auf einem 80spaltigen Bildschirm entspricht eine Programmzeile daher der Länge einer Bildschirmzeile. Auf einem 40spaltigen Bildschirm darf eine Programmzeile zwei Bildschirmzeilen lang sein.

Ist eine Programmzeile kürzer als 80 Zeichen, dann wird sie durch Eingabe der RETURN-Taste abgeschlossen. Ist eine Programmzeile genau 80 Zeichen lang, dann sollte der CBM-Computer von sich aus diese Zeile mit RETURN abschließen, sobald das 80ste Zeichen eingegeben wurde. Sie sollten sich aber nicht auf dieses Erlebnis einlassen; manch ein CBM-Computer läßt Sie Programmzeilen über das 80ste Zeichen hinaus eingeben, die aber beim anschließenden Programmlauf nicht richtig ausgeführt werden. Zur Sicherheit sollten Sie jede Programmzeile vor dem 80sten Zeichen beenden und durch Eingabe von RETURN abschließen.

In einer Programmzeile können mehr als eine Programmanweisung stehen, vorausgesetzt, die Gesamtlänge der Programmzeile ist kürzer als 80 Zeichen. Die Be-

schränkung auf 80 Zeichen und die Darstellung mehrerer Anweisungen in einer Programmzeile gilt für den direkten Dialog und für ablauffähige Programme.

DIREKTER DIALOG UND 1-ZEILIGE PROGRAMME

In direktem Dialog mit dem CBM-Computer kann ein Programm höchstens die Länge einer Bildschirmzeile haben, da alle Anweisungen einer Bildschirmzeile bei Eingabe von RETURN sofort ausgeführt werden. Eine einzelne Bildschirmzeile kann jedoch mehr als eine Anweisung enthalten, so daß einige nützliche Programme in direktem Dialog möglich sind. Wir wollen einige dieser Möglichkeiten untersuchen.

Ein Fragezeichen am Anfang der Eingabe veranlaßt den CBM-Computer, diese Eingabe auf dem Bildschirm darzustellen; das Fragezeichen ist eine abgekürzte Form der BASIC-Anweisung PRINT. Obwohl all unsere Anweisungen in dem oben genannten Beispiel für direkten Dialog mit einem Fragezeichen beginnen, besteht hierfür keine Notwendigkeit. Betrachten Sie das folgende Beispiel:

```
A=π*2

READY.
?A
 6.28318531
```

Der direkte Dialog besteht hier aus zwei Anweisungen. Wenn Sie die erste Anweisung eingeben, wird das Ergebnis nicht auf dem Bildschirm dargestellt, da die Anweisung nicht mit einem ? beginnt; die angegebene Rechnung wird aber ausgeführt. Das Ergebnis wird erst mit der zweiten Anweisung, ?A, dargestellt.

Werden Anweisungen auf einer Programmzeile zusammengefaßt, dann muß jede Anweisung von der nächsten durch einen Doppelpunkt (:) getrennt werden. Die beiden Anweisungen:

```
A=π*2
?A
```

können wie folgt in eine Programmzeile zusammengefaßt werden:

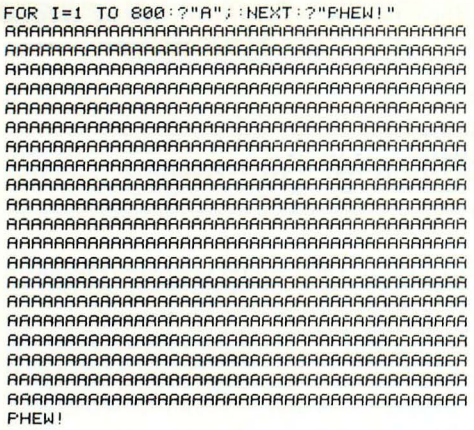
```
A=π*2:?A
```

Beide Anweisungen sind zu einem einzigen, in direktem Dialog ausführbaren Programm geworden.

Da eine Programmzeile aus bis zu 80 Zeichen bestehen darf, können Sie eine Menge Anweisungen in einer Zeile unterbringen und in direktem Dialog ausführen. Betrachten Sie folgendes Beispiel:

```
FOR I=1 TO 800:?"A";:NEXT:?"PHEW!"
```

Machen Sie die Tasteneingaben für dieses Mini-Programm, ohne sich um seine Bedeutung zu kümmern. Wenn Sie die Eingabe mit der Taste RETURN abschließen, wird der Buchstabe A über 20 Bildschirmzeilen eines 40spaltigen Bildschirms geschrieben und auf der 21. Zeile mit dem Ausruf PFUI! abgeschlossen:



Die Programmzeile blieb am Bildschirmanfang stehen. Das Programm hat gerade so viele Bildschirmzeilen beschrieben, daß die Programmzeile bis zum Bildschirmanfang aufrücken mußte, aber nicht aus dem Bildschirm verschwand.

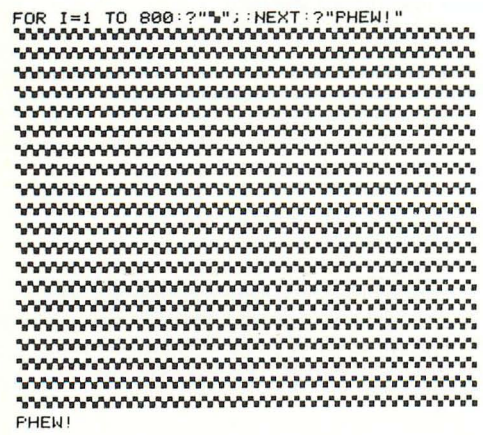
Bei einem 80spaltigen Bildschirm beschreibt das Programm 10 Zeilen mit dem Buchstaben A.

PROGRAMMWIEDERHOLUNG BEI DIREKTEM DIALOG

Nach Ablauf des eben beschriebenen 1-zeiligen Programms in direktem Dialog wird die Nachricht READY dargestellt und der Kursor bleibt auf der untersten Bildschirmzeile stehen.

Eine wichtige Eigenschaft von CBM-BASIC ist, daß die auf dem Bildschirm dargestellte Programmzeile noch lebendig ist. Solange die Programmzeile noch auf dem Bildschirm sichtbar ist, können Sie sie abändern, und den Ablauf des Mini-Programms wiederholen.

Hier ein Beispiel: Bewegen Sie den Kursor nach oben mit der Taste CURSOR UP, bis er auf dem F in FOR steht. Bewegen Sie den Kursor um 15 Stellen nach rechts mit der Taste CURSOR→, bis er auf dem Buchstaben A steht. Drücken Sie eine Taste für graphische Symbole, zum Beispiel die SHIFT-Taste zusammen mit der Taste für das Fragezeichen. Danach geben Sie RETURN ein, und das neue Symbol überschreibt und ersetzt alle A's auf dem Bildschirm. Am Ende ruht der Kursor wieder am Zeilenanfang der unteren Bildschirmzeile:



ÄNDERUNG EIN-ZEILIGER PROGRAMME

Ehe wir mit unserem Programmbeispiel weitere Zeichen auf dem Bildschirm darstellen, ändern wir die Programmzeile derart, daß die Eingabe neuer Zeichen leichter wird. Die veränderte Programmzeile, benutzt für die Darstellung des Buchstabens W, hat dann folgendes Aussehen:

```
C$="W":FOR I=1 TO 800:?C$;:NEXT:?"PHEW!"
```

Die Änderung, oder Edition der Programmzeile verläuft über folgende Schritte:

1. Führen Sie den Kursor mit der HOME-Taste an den Bildschirmanfang zurück, so daß er auf dem ersten Buchstaben der Programmzeile liegt:

```
FOR I=1 TO 800:?"■";:NEXT:?"PHEW!"
```

2. Drücken Sie die Taste INSERT sieben Mal:

```
FOR I=1 TO 800:?"■";:NEXT:?"PHEW!"
```

3. Geben Sie die sechs Zeichen C$ = "W" ein:

```
C$="W":FOR I=1 TO 800:?"■";:NEXT:?"PHEW!"
```

4. Bewegen Sie dann mit der Taste CURSOR→ den Kursor um 14 Stellen nach rechts zum ersten Anführungszeichen:

```
C$="W":FOR I=1 TO 800:?"■";:NEXT:?"PHEW!"
```

5. Geben Sie die beiden Zeichen C$ ein:

```
C$="W":FOR I=1 TO 800:?C$";:NEXT:?"PHEW!"
```

6. Löschen Sie das zweite Anführungszeichen, indem Sie zunächst die Taste CURSOR→ drücken:

```
C$="W":FOR I=1 TO 800:?C$";:NEXT:?"PHEW!"
```

gefolgt von der Tasteneingabe DELETE:

```
C$="W":FOR I=1 TO 800:?C$;:NEXT:?"PHEW!"
```

Alle Änderungen sind ausgeführt; drücken Sie jetzt die Taste RETURN, um den neuen Buchstaben darstellen zu lassen. Danach können Sie den Kursor mit der Taste HOME wieder an den Bildanfang bewegen, um vier Positionen nach rechts bewegen und ein neues Zeichen eingeben. Besonders interessant sind die graphischen Symbole.

ZWISCHENRÄUME IN MEHR-ZEILIGEN PROGRAMMEN

Kämpfen Sie mit der Frage, wo man sich Leerstellen in der Programmzeile erlauben darf und wo nicht? Keine Sorge. CBM BASIC interpretiert nur die Zeichen in einer Programmzeile, Leerstellen werden ignoriert. Ein Beispiel: Die Programmzeile

```
120 FOR I=1 TO 210
```

kann so:

```
120 FOR I=1 TO210
```

oder so

```
120 FORI=1TO210
```

geschrieben werden.

Leerstellen dürfen Sie verwenden, wo Sie die Leserlichkeit einer Programmzeile verbessern. Lediglich BASIC-Anweisungen dürfen nicht durch Leerstellen zerrissen werden. Nach einem Anführungszeichen müssen Sie selbstverständlich dort Leerstellen verwenden, wo ein auszudruckender Text es verlangt.

ABC DER PROGRAMMIERSPRACHE CBM BASIC

Bei der Eingabe von Programmanweisungen sind Rechtschreibregeln einzuhalten, die, alle zusammen genommen, "SYNTAX" genannt werden.

Wie bei den vielen Völkersprachen gibt es verschieden formulierte Programmanweisungen und Rechtschreibungen. Jede Programmiersprache besteht aus eigenen Programmanweisungen und einer eigenen Rechtschreibung. CBM-Computer benutzen nur eine Programmiersprache; sie wird CBM BASIC genannt. Die in diesem Buch beschriebene Syntax gilt nur für die Programmiersprache CBM BASIC. Wie bei den lebendigen Sprachen, gibt es verschiedene Programmiersprachen. Außer BASIC sind die Programmiersprachen PASCAL, FORTRAN, COBOL, APL, PL/M, PL-1 und FORTH weitverbreitet. Die Zahl weniger verbreiteter Programmiersprachen geht in die Hunderte.

Wie lebendige Sprachen haben auch Programmiersprachen ihre Dialekte. Ein in CBM BASIC geschriebenes Programm für Ihren CBM-Computer läuft nicht unbedingt auf einem anderen Computer ab, selbst wenn dieser andere Computer behauptet, BASIC zu verstehen. Dialekte drücken sich in kleineren Abweichungen der Syntax aus, die bei verschiedenen Computern zu beachten sind. Wie Sie jedoch wenig Schwierigkeiten haben, Hochdeutsch zu sprechen, nachdem Sie Bayrisch gelernt haben, werden Sie auch wenig Schwierigkeit haben, nach dem Erlernen von CBM BASIC die BASIC-Sprache eines anderen Computers zu erlernen.

Einige der Rechtschreibregeln sind offensichtlich: Die Additions- und Subtraktions-Beispiele am Anfang dieses Kapitels benutzten eine offensichtliche Syntax. Für diese beiden Anweisungen mußten Sie nicht Programmierer sein, um sie zu verstehen. Die meisten Rechtschreibregeln sind jedoch äußerst willkürlich; ihre Form hat wenig zu tun mit ihrem Sinn. Sie sollten nicht versuchen, Rechtschreibregeln zu erklären; gewöhnlich gibt es keine sinnfällige Erklärung für sie. Ein Beispiel: Warum soll die Multiplikation durch das Symbol (*) dargestellt werden? Von der Schule sind wir ein (x) oder ein (·) gewohnt; aber der Computer hätte keine Möglichkeit, zwischen einem Buchstaben X oder Dezimalpunkt zu unterscheiden. Daher haben sich fast alle Programmiersprachen für das Stern-Symbol (*) zur Darstellung der Multiplikation entschieden. Die Division wird einhellig durch das Symbol (/) dargestellt. Auch hierfür gibt es keine Erklärung; das Standardzeichen (÷) ist nicht auf dem Tastenfeld eines Computers oder einer Schreibmaschine vorhanden, so daß ein anderes Symbol gewählt werden mußte. Zur Syntax von BASIC-Anweisungen gehören außerdem Zeilenzahlen, Datendarstellungen und die eigentlichen Anweisungen an den Computer. Wir werden all das im folgenden darstellen.

ZEILENNUMMERN

Wie wir schon anfangs sagten, muß in einem BASIC-Programm jede Programmzeile eine eigene Zeilennummer haben. Mehr noch: Die erste Zeile eines BASIC-Programms muß die kleinste Zeilennummer tragen, während die letzte Programmzeile eines BASIC-Programms die höchste Zeilennummer trägt. Dazwischen müssen Zeilennummern in aufsteigender Folge vergeben sein. Der CBM-Computer verlangt dies von Ihnen: Unabhängig von der Gelegenheit, bei der Sie eine Programmzeile auf dem Bildschirm eingeben, wird der CBM-Computer diese Programmzeile gemäß Ihrer Zeilennummer in das vorhandene Programm einordnen. Betrachten Sie ein vorhandenes Programm mit den folgenden Zeilennummern:

```
120
130
140
150
160
170
180
190
```

Wenn Sie jetzt auf dem Bildschirm eine neue Anweisung mit der Zeilennummer 165 formulieren, dann kann bei der Eingabe diese neue Anweisung unterhalb der Darstellung des vorhandenen Programms auf dem Bildschirm erscheinen, der CBM-Computer fügt diese Programmzeile jedoch später automatisch zwischen die Zeilennummern 160 und 170 ein. Das folgende Beispiel zeigt diesen Vorgang:

Einfügen einer neuen Programmzeile:
(Bildschirmdarstellung)

	120		120
	130		130
	140		140
	150		150
	160		160
	170	nach erneutem →	165
	180	Auflisten des	170
	190	Programms	180
			190
bei Eingabe →	165		

Wenn die Zeilennummer für eine neue Anweisung die Zeilennummer einer alten Anweisung benutzt, dann wird die alte Anweisung überschrieben durch die neue Anweisung.

In CBM BASIC sind Zeilennummern im Bereich von 1 - 63.999 möglich. Der CBM-Computer interpretiert Zahlen am Anfang einer Programmzeile als Zeilennummer. Erscheinen mehr als fünf Zahlen am Anfang einer Zeile, dann erfolgt eine Fehlermitteilung auf dem Bildschirm: Ein solcher Fehler wird Syntax-Fehler genannt, da Sie dann eine der Rechtschreibregeln von CBM BASIC verletzt haben.

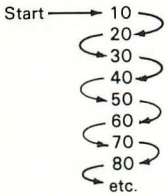
In allen BASIC-Dialekten ist eine aufsteigende Ordnung der Zeilennummern gefordert. Die größte zulässige Zeilennummer jedoch ist bei den verschiedenen BASIC-Dialekten unterschiedlich.

Andere Programmiersprachen außer BASIC verlangen keine Zeilennummer für jede Programmzeile, oder, wenn Zeilennummern vorgeschrieben sind, eine bestimmte aufsteigende Ordnung der Zeilennummern.

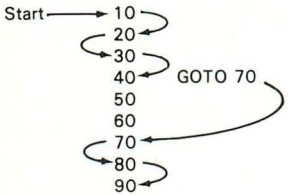
Zeilennummern werden wie Adressen benutzt, die die Stellung der Programmzeile im Programm kennzeichnen. Dieses Konzept ist sehr wichtig, da jedes Programm aus folgenden zwei Typen von Anweisungen besteht:

1. Anweisungen zur Bearbeitung von Daten
2. Anweisungen zur Veränderung der Reihenfolge, in der Daten bearbeitet werden.

Der Gedanke, daß ein Programm vorgeschriebene Operationen in einer starren Reihenfolge ausführt, ist eigentlich sehr einfach. Der Programmablauf beginnt bei der ersten Anweisung im Programm und setzt sich in Reihenfolge der Programmzeilen fort. Hier eine Veranschaulichung dieses Vorgangs:



Wir werden aber bald finden, daß der Ablauf der meisten Programme sich nicht streng an die Reihenfolge der Zeilennummern hält. Hier wird die Bedeutung der Zeilennummern sichtbar: Um die Reihenfolge auszuführender Programmzeilen ändern zu können, brauchen wir Zeilennummern. Im folgenden Beispiel wird die Reihenfolge der Zeilennummern durch einen Sprung von Zeile 40 auf Zeile 70 verändert. (Die hierbei verwendete Anweisung GOTO 70 = GEHE ZU Zeile 70 wird gegen Ende des Kapitels beschrieben):



Zur leichteren Lesbarkeit eines Programms sollte man bei der Eingabe zwischen der Zeilennummer und dem ersten Zeichen einer Anweisung eine oder mehrere Leerstellen lassen. Die Syntax von CBM BASIC macht hierfür keine Vorschriften. Ein Beispiel: Beide der folgenden Programmzeilen sind zulässig:

```
40  GOTO 70
40GOTO 70
```

Hier ein Beispiel für die Verwendung von Leerstellen zur Erhöhung der Lesbarkeit eines Programms:

```
7 0 b b b x
8 0 b b b x
1 0 0 b b x
1 1 0 b b x
1 0 0 0 b x
1 1 0 0 b x
```

b Leerstelle

x erstes Symbol der Anweisung

Wir können nicht ausdrücklich genug die Wichtigkeit unterstreichen, Programme leicht lesbar und leicht verständlich zu schreiben. Das mag wie die Verschwendung von Speicherplatz aussehen, ein Programm enthält aber weniger Fehler und ist leichter zu korrigieren, wenn es einfach zu lesen ist.

DARSTELLUNG VON DATEN

Die Anweisungen nach einer Zeilennummer beschreiben beides: Die vom Computer auszuführende Operation und die Daten, an denen die Operation auszuführen ist. Wir werden jetzt beschreiben, auf welche Weise Daten in CBM BASIC-Programmen dargestellt werden.

Es gibt zwei Arten der Zahlendarstellung in CBM-Computern: Gleitkommazahlen und Ganzzahlen.

GLEITKOMMAZAHLEN

Zahlen werden in der Regel als Gleitkommazahlen im CBM-Computer dargestellt. Bei allen arithmetischen Operationen werden Gleitkommazahlen verwendet. Eine Gleitkommazahl kann eine Ganzzahl sein, oder eine Bruchzahl mit vorangestelltem Dezimalpunkt. Die Zahl kann negativ (–) oder positiv (+) sein. Bei fehlendem Vorzeichen wird angenommen, daß die Zahl positiv ist. Hier einige Beispiele für Gleitkommazahlen, die sich nicht von Ganzzahlen unterscheiden:

```
5
-15
65000
161
0
```

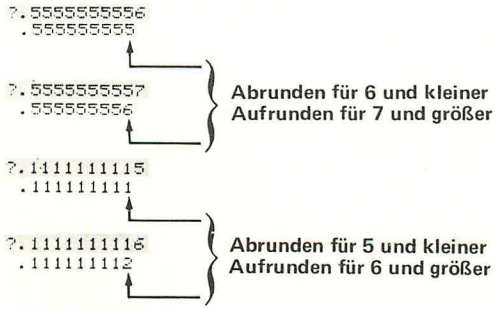
Und hier typische Beispiele für Gleitkommazahlen mit Dezimalpunkt:

```
0.5
0.0165432
-0.0000009
1.6
24.0055
-64.2
3.1416
```

Beachten Sie, daß der Computer einen Syntax-Fehler (SYNTAX ERROR) meldet, wenn Sie in der Zahlendarstellung Kommas verwenden. Schreiben Sie also 65000 und nicht 65,000.

ABRUNDUNG UND AUFRUNDUNG

Zahlen werden mit wenigstens acht Stellen Genauigkeit dargestellt; je nach Zahl kann die Genauigkeit auch neun Stellen betragen. In CBM BASIC wird ab der zehnten Stelle gerundet. Normalerweise erfolgt eine Aufrundung, wenn die zehnte Stelle den Wert 5 oder größer hat, und eine Abrundung, wenn die zehnte Stelle den Wert 4 oder kleiner hat. Manchmal aber gibt es Rundungs-Überraschungen, wie die folgenden Beispiele zeigen:



WISSENSCHAFTLICHE ZAHLENDARSTELLUNG

In CBM BASIC werden Zahlen aus dem Bereich 0,01 bis 999,999,999 in gewohnter Weise dargestellt. Für Zahlen außerhalb dieses Bereiches wird die wissenschaftliche Zahlendarstellung verwendet. Hier einige Beispiele:

```
READY.
?1111111114
1.11111111E+09

READY.
?1111111115
1.11111112E+09
```

Die wissenschaftliche Zahlendarstellung läßt sich an folgender Form erklären:

Zahl E+ee

mit:

Zahl	Mantisse, d.h. Ziffern, die eine Zahl kennzeichnen, z.B. 125 in 0.000125. Bei fehlendem Dezimalpunkt wird angenommen, daß der Dezimalpunkt rechts von ZAHL steht.
E	Buchstabe E als Abkürzung für "Exponent"
+	Vorzeichen des Exponenten (PLUS oder MINUS)
ee	Exponent, ein- oder zweistellig

Der Exponent gibt an, um wieviele Stellen der Dezimalpunkt von ZAHL nach rechts (positiver Exponent) oder links (negativer Exponent) verschoben werden muß, um ZAHL die richtige Größenordnung zu geben.

Hier einige Beispiele:

Wissenschaftl. Schreibung	Standard-Schreibung
2E1	20
10.5E+4	105000
66E+2	6600
66E-2	0.66
-66E-2	-0.66
1E-10	0.0000000001
94E20	9400000000000000000000

Die wissenschaftliche Zahlendarstellung ermöglicht eine bequeme Schreibung sehr großer oder sehr kleiner Zahlen. Zahlen mit mehr als 10 Stellen werden von CBM BASIC automatisch in wissenschaftlicher Notierung dargestellt:

```
?.009
 9E-03

READY.
?.01
 .01

READY.
?999999998.9
 999999999

READY.
?999999999.6
 1E+09
```

Aber selbst bei wissenschaftlicher Zahlendarstellung gibt es Grenzen der in CBM BASIC darstellbaren Zahlen. Diese Grenzen sind:

Größte Gleitkomma-Zahl: +1.70141183E+38
Kleinste Gleitkomma-Zahl: +2.93873588E-39

Beim Überschreiten der größten Gleitkommazahl meldet sich der Computer mit einem Überlauf-Fehler (OVERFLOW ERROR):

```
?1.70141183E+38
 1.70141183E+38

READY.
?-1.70141183E+38
-1.70141183E+38
```

Zulässiger Zahlenwert

```
READY.
?1.70141184E+38

?OVERFLOW ERROR
READY.
?-1.70141184E+38

?OVERFLOW ERROR
```

Zu großer Zahlenwert: Überlauf-Meldung

Beim Unterschreiten der kleinsten Gleitkommazahl ersetzt der Computer die Zahl durch eine Null:

```
?2.93873588E-39
 2.93873588E-39

READY.
?-2.93873588E-39
-2.93873588E-39
```

Zulässiger Zahlenwert

```
READY.
?2.93873587E-39
 0

READY.
?-2.93873587E-39
 0
```

Zu kleiner Zahlenwert: Ersatz durch Null

GANZZAHLEN

Eine Ganzzahl ist eine Zahl ohne Dezimalpunkt und ohne Bruchanteil. Das Vorzeichen der Ganzzahl kann negativ (–) oder positiv (+) sein; bei fehlendem Vorzeichen wird eine positive Ganzzahl angenommen. Wichtig: Ganzzahlen sind auf den Zahlenbereich – 32.767 bis + 32.767 beschränkt. Hier typische Beispiele für Ganzzahlen:

0
1
44
32699
–15

Jede Ganzzahl kann auch als Gleitkommazahl dargestellt werden, da Ganzzahlen eine Teilmenge der Gleitkommazahlen darstellen. Vor einer Verwendung in arithmetischen Operationen verwandelt CBM BASIC automatisch alle Ganzzahlen in Gleitkommazahlen.

ZEICHENKETTEN

Das Wort "Zeichenkette" wird für Daten gebraucht, die nicht reine Zahlen sind und im CBM-Computer als Text verarbeitet werden. Wir haben für Darstellungen auf dem Bildschirm des CBM-Computers bereits Zeichenketten verwendet.

Eine Zeichenkette besteht aus einem oder mehreren Zeichen, die zwischen zwei Anführungszeichen eingeschlossen sind. Hier einige Beispiele für Zeichenketten:

```
"HI!"
"SYNERGY"
"12345"
"$10.44 IS THE AMOUNT"
"22 UNION SQUARE, SAN FRANCISCO, CA"
```

Eine Zeichenkette zwischen Anführungszeichen kann folgende Zeichen enthalten: Buchstaben des Alphabets, Zahlen, Symbole der Zeichensetzung, Sonderzeizeichen, graphische Symbole und Symbole, mit denen die Betätigung von Kontrolltasten des Kursors und der Tasten für Negativschrift zwischen den Anführungszeichen einer Zeichenkette dargestellt werden. Tabelle 4-1 enthält eine Zusammenstellung der letztgenannten Gruppe von Symbolen. Die einzigen Tasten, die nicht zwischen den Anführungszeichen einer Zeichenkette benutzt werden können, sind die Tasten RUN/STOP, RETURN und INSERT/DELETE.

Alle Zeichen einer Zeichenkette werden auf dem Bildschirm so dargestellt, wie sie in der Zeichenkette stehen. Eine Besonderheit bilden die Kontrolltasten des Kursors und die Tasten für Negativschrift, bei deren Betätigung normalerweise kein Symbol auf dem Bildschirm erscheint; um die Betätigung dieser Tasten in einer Zeichenkette sichtbar zu machen, werden bestimmte Symbole in Negativschrift benutzt, wie die Zusammenstellung in Tabelle 4-1 zeigt.

Zeichenketten werden als Teil einer Anweisung eingegeben. Da Anweisungen in einer Programmzeile aus höchstens 80 Zeichen bestehen dürfen, können über das Tastenfeld nur Zeichenketten mit weniger als 80 Zeichen eingegeben werden; die restlichen Zeichen werden für die Zeilennummer und die Darstellung der Anweisung benötigt.

Im Speicher eines CBM-Computers dürfen Zeichenketten dagegen eine Länge bis zu 255 Zeichen haben. Derart lange Zeichenketten werden durch Verkettung kürzerer Zeichenketten erzeugt. Das Verfahren der Verkettung wird uns später beschäftigen.

VARIABLE

Am Anfang des Kapitels benutzten wir bei der Beschreibung des direkten Dialogs folgende Anweisungen:

```
A=π*2
?A
```

Diese Anweisungen faßten wir dann zu einer Programmzeile zusammen:

```
A=π*2:?A
```

In diesen Programmen stellt der Buchstabe A eine Variable dar.

Die Idee einer Variablen ist leicht zu verstehen. Betrachten Sie folgende Anweisungen:

```
100 A=B+C
200 ?A
```

Diese beiden Anweisungen stellen die Summe zweier Zahlen auf dem Bildschirm dar. Aber welche Zahlen werden summiert? Es sind diejenigen Zahlen, die im Augenblick der Programmausführung von den Variablen B und C dargestellt werden. Im folgenden Beispiel:

```
90 B=4.65
95 C=3.72
100 A=B+C
200 ?A
```

wird vor Programmausführung der Variablen B der Wert 4.65 zugeordnet, während die Variable C den Wert 3.72 erhält. Die Summe A ist deshalb 8.37.

Tabelle 4-1. Darstellung von Funktionstasten in Zeichenketten

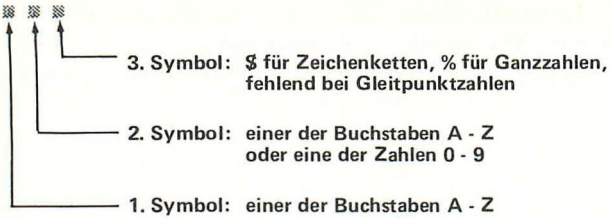
Funktion	Tasteneingabe	Graph. Darstellung *
Negativschrift EIN	OFF RVS ON	(R in Negativschrift)
Negativschrift AUS	SHIFT + OFF RVS ON	(w. o., jedoch SHIFT + R)
Kursor an Bildschirmanfang	CLR SCREEN HOME	(S in Negativschrift)
Bildschirm löschen	SHIFT + CLR SCREEN HOME	(w.o., jedoch SHIFT + S)
Kursor ↓	CURSOR	(Q in Negativschrift)
Kursor ↑	SHIFT + CURSOR	(w.o., jedoch SHIFT + Q)
Kursor →	CURSOR	(J in Negativschrift)
Kursor ←	SHIFT + CURSOR	(w.o., jedoch SHIFT + J)

* Tasteneingabe der Funktion nach dem ersten Anführungszeichen in z.B. PRINT "....." führt zur angegebenen graphischen Darstellung.

Variable können zur Darstellung von Zeichenketten oder Zahlen verwendet werden. Der Umgang mit Variablen ist Ihnen sicherlich von der Schulzeit bekannt. Der Name einer Variablen kann anschaulich verglichen werden mit dem Namen auf einem Briefkasten: Jeder Einwurf in den Briefkasten wird dem Namen auf dem Briefkasten zugeordnet. Bei Variablen sagt man, ein Wert wird der Variablen zugewiesen.

NAMEN VON VARIABLEN

Der Name einer Variablen kann aus einem, zwei oder drei Zeichen bestehen. Für Stellung und Typ der Zeichen gilt folgende Regel:



Das letzte Zeichen im Namen einer Variablen teilt also mit, welchen Typ von Daten die Variable darstellen soll.

Beachten Sie, daß für die erste und zweite Stelle im Namen der Variablen nur Buchstaben benutzt werden dürfen, bei deren Eingabe die Umschalttaste (SHIFT) nicht betätigt wurde. Je nach Modell des CBM-Computers können dies Großbuchstaben oder Kleinbuchstaben sein.

Gleitkomma-Variable sind die am häufigsten benutzten in CBM BASIC. Hier sind Beispiele für Namen von Gleitkomma-Variablen:

```
A
B
C
A1
AA
Z5
```

Und hier Beispiele für Namen von Ganzzahl-Variablen:

```
A%
B%
C%
A1%
MN%
X4%
```

Erinnern Sie sich: Auch Ganzzahlen können durch Gleitkomma-Variable dargestellt werden.

Schließlich Beispiele für die Namen von Zeichenketten-Variablen:

```
A$
M$
MN$
M1$
ZX$
F6$
```

Der Name einer Variablen kann aus mehr als zwei alphanumerischen Zeichen bestehen, von CBM BASIC werden aber nur die ersten beiden Zeichen als Name gezählt. Die beiden Namen BANANE und BANDAGE werden daher als gleiche Namen interpretiert, da beide mit BA beginnen. In CBM BASIC sind Variablen-Namen mit bis zu 255 Zeichen gestattet. Hier einige Beispiele für Namen von Variablen mit mehr als zwei Zeichen:

MAGIC$	gelesen als	MA$
N123456789		N1
MMM$		MM$
ABCDEF%		AB%
CALENDAR		CA

Beachten Sie bei der Benutzung von Variablen-Namen von mehr als zwei Zeichenlängen die folgenden Hinweise:

1. Nur die ersten beiden Zeichen und das Symbol für den Datentyp ($ oder %) zählen. Verwenden Sie nicht lange Namen wie LOOP 1 und LOOP 2; beide Namen werden gleich interpretiert, als LO.
2. CBM BASIC enthält eine Reihe "reservierter Worte", die eine Bedeutung in BASIC-Anweisungen haben. Im Namen einer Variablen darf nirgends ein reserviertes Wort eingebettet vorkommen. Tabelle 4-4 enthält eine Zusammenstellung reservierter Worte.
3. Zusätzliche Zeichen im Namen einer Variablen belegen zusätzlichen Speicherplatz, den Sie vielleicht bei längeren Programmen nötig haben. Der Vorteil langer Variablen-Namen liegt in der leichteren Lesbarkeit eines Programms. Der Name TEIL#, beispielsweise, ist in einem Inventur-Programm sinnvoller als TE.

OPERATOREN

Die BASIC-Anweisung:

```
100 ?10.2+4.7
```

teilt dem CBM-Computer mit, die Zahlen 10.2 und 4.7 zu addieren und das Ergebnis auf dem Bildschirm darzustellen. Die Anweisung:

```
250 C=A+B
```

teilt dem CBM-Computer mit, die beiden durch Variable dargestellten Gleitkommazahlen zu addieren und die Summe der Gleitkomma-Variablen C zuzuweisen.

In diesen Beispielen stellt das Pluszeichen (+) einen Operator dar, d.h. den Operator für eine Addition.

Arithmetische Operatoren für Addition, Subtraktion, Multiplikation und Division sind uns aus der Schulzeit vertraut. Außerdem gibt es noch zwei andere Typen von Operatoren: vergleichende Operatoren und logische Operatoren (auch Boole'sche Operatoren). An sich sind auch diese Operatoren leicht zu verstehen, aber sie erfordern etwas mehr Erklärung, da sie uns nicht vom täglichen Gebrauch vertraut sind.

Tabelle 4-2 enthält eine Zusammenstellung der Operatoren in BASIC. Wir werden jede Gruppe von Operatoren der Reihe nach betrachten, beginnend mit den arithmetischen Operatoren.

Tabelle 4-2. Operatoren in BASIC

	Rangordnung	Operator	Bedeutung
	hoch 9	( )	Angabe der Reihenfolge auszuführender Operationen
Arithmetische Operatoren	8 7 6 6 5 5	↑ – * / + –	Exponentialbildung Negative Zahl Multiplikation Division Addition Subtraktion
Vergleichende Operatoren	4 4 4 4 4 4	= <> < > <= oder= < >=oder= >	gleich ungleich kleiner als größer als kleiner oder gleich größer oder gleich
Logische Operatoren	3 2 1 niedrig	NOT AND OR	logische Verneinung logisches UND logisches ODER

ARITHMETISCHE OPERATOREN

Mit einem arithmetischen Operator werden Addition, Subtraktion, Multiplikation, Division oder Expotentialbildung symbolisiert. Zur Ausführung arithmetischer Operationen werden Gleitkommazahlen verwendet. Ganzzahlen werden automatisch in Gleitkommazahlen umgewandelt, ehe sie in arithmetischen Operationen verwendet werden. Das Ergebnis einer arithmetischen Operation wird automatisch rückverwandelt in eine Ganzzahl, wenn eine Ganzzahl-Variable zur Darstellung des Ergebnisses angegeben wurde.

Man nennt Daten, die durch einen Operator verknüpft werden sollen, "Operanden". Jeder arithmetische Operator erfordert zwei Operanden, die Zahlen und/oder numerische Variable sein können.

ADDITION (+)

Das Pluszeichen gibt an, daß der Operand auf der linken Seite des +-Zeichens zum Operanden auf der rechten Seite addiert werden muß. Für numerische Daten bedeutet dies eine einfache Addition:

```
2+2
A+B+C
X%+1
BR+10E-2
```

Das Pluszeichen wird aber auch benutzt, um Zeichenketten zu "addieren" und längere Zeichenketten zu bilden. Diese Operation nennt man Verkettung. Den Unterschied zwischen der Addition von Zahlen und der Addition von Zeichenketten kann man wie folgt veranschaulichen:

Addition von Zahlen:
ZAHL 1 + ZAHL 2 = ZAHL 3

Addition von Zeichenketten:
ZEICHENKETTE 1 + ZEICHENKETTE 2 = ZEICHENKETTE1 ZEICHENKETTE2

Durch Verkettung können Zeichenketten mit bis zu 255 Zeichen gebildet werden. Beispiele hierfür sind:

"FOR"+"WARD"	=	"FORWARD"
"HI"+" "+"THERE"	=	"HI THERE"
A$+B$	=	Verkettung der stellvertretend durch A$, B$ dargestellten Zeichenketten
"1" + CH$+E$	=	Verkettung einer 1 mit den stellvertretend durch CH$, E$ dargestellten Zeichenketten

Werden in diesem Beispiel den Variablen A$ und B$ die Zeichenketten "FOR" und "WARD" zugewiesen, dann würde A$ + B$ zum gleichen Ergebnis führen wie "FOR" + "WARD".

SUBTRAKTION (–)

Ein Minuszeichen schreibt vor, daß der Operand auf der rechten Seite des Minuszeichens vom Operanden auf der linken Seite zu subtrahieren ist, wie folgende Beispiele zeigen:

Operation	Ergebnis
4 - 1	3
100 - 64	36
A - B	Subtraktion der durch Variable B dargestellten Zahl von der durch Variable A dargestellten Zahl
55 - 142	–87

Werden im Beispiel oben den Variablen A und B die Werte 100 und 64 zugeordnet, dann stellen das zweite und dritte Beispiel identische Operationen dar.

Das Minuszeichen dient auch dazu, negative Zahlen zu kennzeichnen, wie diese Beispiele zeigen:

```
-5
-9E4
-B
4--2
```

Beachte: 4– –2 ist das gleiche wie 4 + 2

MULTIPLIKATION (*)

Ein Sternchensymbol schreibt vor, daß der Operand auf der rechten Seite des Sternsymbols zu multiplizieren ist mit dem Operanden auf der linken Seite, wie folgende Beispiele zeigen:

Operation	Ergebnis
100*2	200
50*0	0
A*X1	Multiplikation der beiden durch A und X1 dargestellten Gleitkommazahlen
R%*14	Multiplikation einer mit R% dargestellten Ganzzahl mit der Zahl 14

Wenn man in den Beispielen oben der Variablen A und X1 die Zahlenwerte 4.2 und 9.63 zuordnet, dann erzeugt die angegebene Operation das Ergebnis 40.446. Zur Nachbildung der ersten Operation müßten A und X1 die Zahlenwerte 100 und 2 zugewiesen werden; da A und X1 Gleitkomma-Variable sind, würden diese beiden Zah-

len in der Form 100.0 und 2.0 gespeichert werden. Für die Darstellung der Zahlen 100 und 2 als Ganzzahlen müßten die Variablen A% und X1% lauten.

DIVISION (/)

Ein Schrägstrich schreibt als Operation vor, daß ein Operand auf der linken Seite des Schrägstrichs zu dividieren ist mit dem Operanden auf der rechten Seite, wie folgende Beispiele zeigen:

Operation	Ergebnis
10/2	5
6400/4	1600
A/B	Division der durch Variable A dargestellten Gleitkommazahl durch die mit Variable B dargestellte Gleitkommazahl
4E2/XR	Division der Zahl 400 durch die mit Variable XR dargestellte Gleitkommazahl

Das dritte Beispiel, A/B, kann die gleiche Operation wie im ersten und zweiten Beispiel darstellen, obwohl A und B Gleitkomma-Variable sind. Ganzzahlen würden dann in Form von Gleitkommazahlen gespeichert werden. Eine genaue Nachbildung der ersten beiden Operationen wäre A%/B%.

EXPONENTIALBILDUNG (↑)

Das Pfeilsymbol schreibt als Operation vor, daß der Operand auf der linken Seite des Pfeiles in die auf der rechten Seite des Pfeils angegebene Potenz erhoben wird. Ist der Operand rechts des Pfeils eine 2, dann wird die Zahl links des Pfeils quadriert. Als Exponent können Zahlen, Variable oder Ausdrücke auftreten, sofern die Exponentialbildung eine Zahl im zulässigen Zahlenbereich der Gleitkommazahlen ergibt. Hier einige Beispiele:

Operation	Ergebnis
2↑2	4
12↑2	144
1↑3	1
A↑5	die Gleitkommazahl A wird nach Zuweisung eines Werts in die 5te Potenz erhoben
2↑6,4	84.4485064
NM↑-10	nach Zuweisung eines Werts wird die Gleitkommazahl NM in die Potenz −10 erhoben
14↑F	14 erhoben zur Potenz F nach Zuweisung eines Werts zur Gleitkommavariablen F

RANGORDNUNG DER OPERATOREN

Ein Ausdruck kann mehrere arithmetische Operatoren enthalten, wie das folgende Beispiel zeigt:

```
A+C*10/2↑2
```

In solchen Fällen gibt es eine feste Rangordnung, nach der die Operationen ausgeführt werden (s.a. Tabelle 4-2, Spalte "Rangordnung"). Zuerst wird die Expo-

nentialbildung (↑), dann die Auswertung des Vorzeichens, danach Multiplikation und Division (*, /), schließlich Addition und Subtraktion (+, –) ausgewertet. Operatoren der gleichen Rangordnung in einem Ausdruck werden von links nach rechts ausgewertet. Diese Rangordnung der Operatoren kann durch Verwendung von Klammern verändert werden. Operationen in Klammerausdrücken werden zuerst ausgeführt. Hier einige Beispiele:

Operation	Ergebnis
4+1•2	6
(4+1)•2	10
100•4/2–1	199
100•(4/2–1)	100
100•(4/(2–1))	400

Bei Klammerausdrücken wertet CBM BASIC zuerst die innersten Klammern aus. Verschachtelungen von Klammerausdrücken können ohne Beschränkung verwendet werden, um die Rangfolge der Operationen in einem Ausdruck klarzustellen.

VERGLEICHS-OPERATOREN

Vergleichende Operatoren beantworten das Ergebnis folgender Vergleiche mit wahr oder falsch: Größer als (>), kleiner als (<), gleich (=), ungleich (< >), größer als oder gleich (> =) und kleiner als oder gleich (< =). Hier einige Beispiele:

Operation	Ergebnis
1=5–4	wahr (–1)
14>66	falsch (0)
15>=15	wahr (–1)
A<>B	abhängig von den Wertzuweisungen zu A, B

In CBM BASIC werden die Antworten wahr oder falsch einer Vergleichsoperation in Form der Zahlen 0 (für wahr) und –1 (für falsch) dargestellt. Diese Zahlen 0 und –1 können dann in Gleichungen verwendet werden. Ein Beispiel: In dem Ausdruck (1=1) * 4 ist (1=1) wahr. Die Antwort "wahr" entspricht –1, so daß der Ausdruck das gleiche ist wie (–1) *4= –4. Sie können jeden Vergleichsoperator in einem CBM BASIC-Ausdruck verwenden, wie folgende Beispiele zeigen:

25+(14>66)	entspricht	25+0
(A+(1=5–4))•(15>=15)	entspricht	(A–1)•(–1)

Vergleichs-Operatoren können auch zum Vergleich von Zeichenketten verwendet werden. Bei Vergleichen wird für die Buchstaben des Alphabets eine Rangordnung angenommen, die Ähnlichkeit mit der Rangordnung von Zahlen hat: A<B, B<C, C<D, usw. Zeichenketten werden zeichenweise verglichen, beginnend mit den linksstehenden Zeichen. Hier einige Beispiele:

Operation	Ergebnis
"A"<"B"	wahr (–1)
"X"="XX"	falsch (0)
C$=A$+B$	das Ergebnis hängt von den Textzuweisungen zu C$, A$, B$ ab

Vergleichs-Operationen von Zeichenketten liefern wie bei Zahlen die Antworten "wahr" und "falsch" in Form der Zahlenwerte –1 (für wahr) und 0 (falsch). Hier einige Beispiele:

("JONES">"DOE")+37 entspricht -1+37
("AAA"<"AA")•(Z9-("OTTER">"AB")) entspricht 0•(Z9-(-1))

LOGISCHE (BOOLE'SCHE) OPERATOREN

Logische Operatoren befähigen ein Programm zu logischen Entscheidungen. Die vier logischen Grund-Operatoren sind: AND, OR, EXCLUSIVE OR und NOT. In CBM BASIC stehen Ihnen drei dieser Operatoren zur Verfügung: AND, OR und NOT.

Wenn Sie kein Gefühl für logische Operatoren haben, dann kann Ihnen das folgende Beispiel aus dem täglichen Leben den Sinn logischer Operatoren veranschaulichen.

Nehmen Sie an, Sie sind mit zwei Kindern beim Bäcker zum Einkaufen von Semmeln, und müssen sich für eine der vielen Formen entscheiden.

Der logische Operator AND besagt, daß die Entscheidung für eine Semmel gefallen ist, wenn Kind A UND Kind B sich auf die gleiche Form geeinigt haben.

Der logische Operator OR besagt, daß der Einkauf ohne Streit verläuft; gekauft wird, was entweder Kind A ODER Kind B sagt.

Der logische Operator NOT besagt, daß es schon vor dem Einkauf Streit gab. Die Entscheidung von Kind B ist immer das Gegenteil – das NICHT – der Entscheidung von Kind A.

Die eben behandelten Aussagen der Kinder werden im Computer durch Zahlen dargestellt: Das Vorhandensein einer Aussage durch 1, die Abwesenheit einer Aussage durch 0. Tabelle 4-3 gibt eine Zusammenstellung der in CBM-Computern möglichen logischen Operatoren; derartige Tabellen werden gewöhnlich "Boole'sche Wahrheitstabelle" genannt, nach dem Namen des Mathematikers BOOLE.

Logische Operatoren dienen zur Beeinflussung der Ablauflogik von Programmen; hier sind Beispiele dafür:

```
10 IF A=100 AND B=100 GOTO 30
```
: Verzweigen auf Zeile 30, wenn A und B den Wert 100 haben

```
20 IF X<Y AND B>=44 THEN F=0
```
: Variable F auf Null setzen, wenn X kleiner als Y und B größer oder gleich 44 sind

```
30 IF A=100 OR B=100 GOTO 20
```
: Verzweigen auf Zeile 20, wenn A oder B den Wert 100 haben

```
40 IF X<Y OR B>=44 THEN F=0
```
: Variable F auf Null setzen, wenn X kleiner als Y oder wenn B größer oder gleich 44 ist

```
50 IF A=1 AND B=2 OR C=3 GOTO 40
```
: Verzweigen auf Zeile 40, wenn (A=1 und B=2) oder wenn (C=3)

Mit logischen Operatoren kann auch geprüft werden, ob eine Variable einen Zahlenwert darstellt oder Null ist. Die Anweisung IF A wird vom Computer als Aufforderung zu einer Vergleichsoperation verstanden, wie das Beispiel unten zeigt. Besitzt die Variable A einen Zahlenwert (wahr), dann wird die Zuweisung B=2 ausgeführt:

```
IF A     THEN B=2
IF A<>0  THEN B=2
```
Beide Anweisungen sind gleichwertig

Verzweigen, wenn B falsch ist, d.h. B=0.

```
IF NOT B GOTO 100
IF B=0   GOTO 100
```
Die zweite Anweisung drückt diese Bedingung direkter aus.

Alle logischen Operatoren arbeiten mit Ganzzahlen. Wenn Sie logische Operatoren auf Gleitkommazahlen anwenden, dann werden diese Zahlen automatisch in Ganzzahlen umgewandelt; Gleitkommazahlen müssen also im zulässigen Zahlenbereich von Ganzzahlen liegen.

Logische Operatoren können nicht auf Zeichenketten angewandt werden.

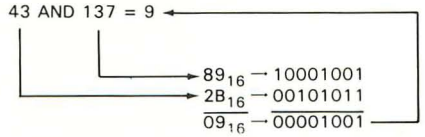
Am Anfang Ihrer Programmierpraxis werden Sie logische Operatoren, wie sie hier beschrieben sind, wahrscheinlich nicht benötigen. Wenn Sie ein ungutes Gefühl über dieses Thema haben, gehen Sie einfach zum nächsten Abschnitt weiter.

Tabelle 4-3. Boole'sche Wahrheitstabelle

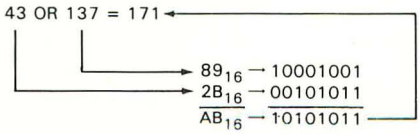
A AND B	1 AND 1 = 1 0 AND 1 = 0 1 AND 0 = 0 0 AND 0 = 0
A OR B	1 OR 1 = 1 0 OR 1 = 1 1 OR 0 = 1 0 OR 0 = 0
NOT A	NOT 1 = 0 NOT 0 = 1

Wie die Wahrheitstabelle 4-3 zeigte, arbeiten logische Operatoren nur mit den Zahlen 0 und 1. Da im CBM-Computer alle Zahlen binär dargestellt werden, d.h. als eine Folge von 0 und 1, können logische Operatoren auf beliebige Zahlen angewandt werden. Werden daher zusammen mit einem logischen Operator Dezimalzahlen eingegeben, dann erfolgt im CBM-Computer eine Umwandlung in Binärzahlen, ehe die logischen Operationen beginnen. Negative Dezimalzahlen werden binär im 2er-Kompliment dargestellt.

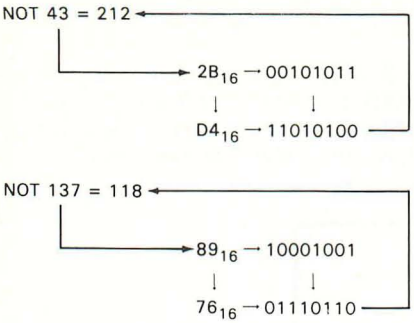
Eine AND-Operation nimmt daher folgenden Verlauf:



Für das gleiche Zahlenbeispiel ergibt sich bei einer OR-Operation folgendes:



Schließlich das Beispiel einer NOT-Operation:



Logische Operationen dieser Art finden Sie in technischen Anwendungen.
Wird ein logischer Operator auf Ergebnisse von Vergleichsoperationen angewandt, wie das folgende Beispiel für die Vergleichsoperatoren (=, <) zeigt:

```
A=1 OR C<2
```

dann ist zu beachten, daß Vergleichsoperationen nur −1 oder 0 als Ergebnis liefern, worauf die logische Operation angewandt wird:

```
(−1 bzw. 0) OR (−1 bzw. 0)
```

Im CBM-Computer wird −1 als Binärzahl 1111 dargestellt (das Beispiel beschränkt sich auf vierstellige Binärzahlen), so daß eine logische Operation in Verbindung mit Vergleichsoperationen tatsächlich auf folgende Aussagen angewandt wird:

```
(1111 bzw. 0000) OR (1111 bzw. 0000)
```

Betrachten wir die noch verwickeltere Operation:

```
IF A=B AND C<D GOTO 40
```

Zuerst werden Vergleichsoperationen ausgeführt. Nehmen wir an, die erste Vergleichsoperation liefert als Ergebnis −1 (wahr) und die zweite Operation 0 (falsch). Der logische Operator AND ist dann auf folgende Aussagen anzuwenden:

```
IF −1 AND 0 GOTO 40
```

Das Ergebnis von 1111 AND 0000 ist 0:

```
IF 0 GOTO 40
```

Erinnern Sie sich, daß bei einem Ausdruck der Form IF 0 die Vergleichsoperation <> 0 ausgeführt wird:

```
IF 0 <> 0 GOTO 40
```

Die Verzweigung GOTO findet also nicht statt. Im Gegensatz zu diesem Beispiel kann die Anwendung logischer Operationen auf zwei Variablen beliebige Ganzzahlen ergeben:

```
IF A% AND B% GOTO 40
```

Es sei A% = 255 und B% = 240. Die logische Operation 255 AND 240 ergibt 240. Die Anweisung nimmt daher folgende Form an:

```
IF 240 GOTO 40
```

oder, bei Darstellung der stattfindenden Vergleichsoperation:

```
IF 240 <> 0 GOTO 40
```

In diesem Fall wird die Verzweigung GOTO ausgeführt.
Vergleichen Sie schließlich noch folgende Zuweisungen:

```
A = A AND 10
A = A <10
```

Im ersten Beispiel wird der augenblickliche Wert von A logisch verknüpft mit 10 und das Ergebnis als neuer Wert in A gespeichert. A muß im Ganzzahlbereich −32767 bis +32767 liegen.

Im zweiten Beispiel wird die Vergleichsoperation A< 10 ausgeführt und das Ergebnis −1 oder 0 als neuer Wert in A gespeichert.

TABELLEN (FELDER)

Die Darstellung von Tabellen wird sehr häufig in jeder Art von Computerprogramm verwendet. Wenn Sie das Anlegen von Tabellen im Computer nicht verstehen, dann müssen Sie es hier lernen. Die jetzt folgenden Informationen sind außerordentlich wichtig für Ihre Programmiererfolge.

Von der Idee her sind Tabellen sehr einfach. Haben Sie mehrere Daten, dann brauchen Sie nicht jedem Datum einen eigenen Namen geben, sondern nur einen Kollektiv-Namen zu vergeben. Die einzelnen Daten werden dann durch eine Platznummer gekennzeichnet, die in der Mathematik und bei der Datenverarbeitung Index oder Dimension genannt wird.

Ein Beispiel: In der Einkaufsliste eines Großmarkts gebe es sechs Eintragungen aus der Abteilung Fleisch und Geflügel, vier aus der Abteilung Früchte und Obst und drei aus der Abteilung Milchprodukte. Sehen Sie jetzt, wie für diese drei Gruppen Kollektiv-Namen und Indizes vergeben wurden:

```
FG$(0)="PUTEN"                  OG$(0)="ORANGEN"
FG$(1)="PUTENSCHNITZEL"         OG$(1)="AEPFEL"
FG$(2)="SUPPENHUEHNER"          OG$(2)="KAROTTEN"
FG$(3)="HUEHNERKEULEN"          OG$(3)="BOHNEN"
FG$(4)="SALAMI"
FG$(5)="WUERSTCHEN"             MP$(0)="MILCH"
                                MP$(1)="SAHNE"
                                MP$(2)="HUETTENKAESE"

ABKUERZUNGEN: FG=FLEISCH/GEFLUEGEL;OG=OBST/GEMUESE;
              MP=MILCHPRODUKTE
```

Wir könnten die Idee einer Tabelle um einen Schritt erweitern, indem wir alle Eintragungen in die Einkaufsliste unter dem kollektiven Variablennamen EL$ zusammenfassen und zwei Indizes verwenden. Der erste Index gibt jetzt die Produktgruppe an, der zweite Index identifiziert das Produkt:

```
EL$(0,0)=FG$(0)    EL$(1,0)=OG$(0)    EL$(2,0)=MP$(0)
EL$(0,1)=FG$(1)    EL$(1,1)=OG$(1)    EL$(2,1)=MP$(1)
EL$(0,2)=FG$(2)    EL$(1,2)=OG$(2)    EL$(2,2)=MP$(3)
EL$(0,3)=FG$(3)    EL$(1,3)=OG$(3)
EL$(0,4)=FG$(4)
EL$(0,5)=FG$(5)
```

In Tabellen können Ganzzahlen, Gleitkommazahlen oder Zeichenketten gespeichert werden; eine einzelne Variable, z.B. EL$ (0,0), kann jedoch jeweils nur einen Datentyp speichern. Mit anderen Worten: Eine einzige Variable kann nicht abwechselnd zum Speichern von Ganzzahlen und Gleitkommazahlen verwendet werden.

Die Darstellung von Tabellen stellt eine nützliche Abkürzung zur Beschreibung großer Datenmengen dar. Ein Beispiel: Eine Zahlentabelle bestehe aus 20 Spalten mit 10 Zahlen in jeder Spalte. Das ergibt 200 Zahlen. Wie hätten Sie gerne die 200 Namen vergeben? Es ist viel einfacher, der ganzen Tabelle nur einen Namen zu geben und die Zahlen durch ihre Stellung in der Tabelle zu identifizieren. Genau das macht eine Tabellen-Darstellung.

Tabellen können einen oder mehrere Indizes (eine oder mehrere Dimensionen) haben. Bei **einem** Index besteht die Tabelle aus nur einer Spalte. (Ingenieure würden das Wort "Vektor" für eine derartige Tabelle verwenden). Mit zwei Indizes kann eine Tabelle aus Spalten und Zeilen angelegt werden: Der eine Index kennzeichnet die Spalten, der andere die Zeilen. Mit drei Indizes kann beispielsweise ein Tabellenwerk dargestellt werden: Je zwei Indizes beschreiben Spalte und Zeile, der dritte Index numeriert die Tabellen. Bei vier und mehr Indizes hat unsere Vorstellungskraft Schwierigkeiten; nicht jedoch der Computer.

Lassen Sie uns die Darstellung von Tabellen genauer untersuchen.

In einer eindimensionalen Tabelle hat jeder Tabelleneintrag folgende Form:

NAME (i)

mit:

NAME	Name der Tabelle, gebildet wie Variablen-Namen (s. a. Tabelle 4-4)
i	Index, Dimension eines Tabellen-Eintrags

Ein Beispiel: Die fünf Tabellen-Einträge einer eindimensionalen Tabelle A kann man sich wie folgt vorstellen:

A (0)
A (1)
A (2)
A (3)
A (4)

Die Anzahl der Tabellen-Einträge ist gleich dem höchsten Index + 1, da die Zählung des Index bei Null beginnt.

In einer 2-dimensionalen Tabelle werden Einträge wie folgt gekennzeichnet:

NAME (i, j)

mit:

NAME	Tabellen-Name (s. o.)
i	Spaltenindex
j	Zeilenindex

Eine 2-dimensionale Tabelle A$ mit zwei Spalten und drei Zeilen je Spalte kann wie folgt vorgestellt werden:

A$ (0,0)	A$ (1,0)
A$ (0,1)	A$ (1,1)
A$ (0,2)	A$ (1,2)

Diese Tabelle besteht aus sechs Einträgen. Die Größe einer Tabelle ist allgemein das Produkt aus größtem Spalten-Index + 1, multipliziert mit dem größten Zeilenindex + 1. Tabellen mit mehreren Indizes sehen wie folgt aus:

NAME (i,j.k, . . .)

Tabellen mit elf oder weniger Einträgen werden vom CBM-Computer unmittelbar angenommen. Tabellen mit mehr als 11 Einträgen müssen am Programmanfang angekündigt oder "erklärt" werden. Hierfür verwendet man Dimensions-Anweisungen, die weiter unten erklärt werden. Der Name einer Tabelle und ein gleichlautender Name einer Variablen werden von CBM BASIC als unterschiedlich erkannt und auch so behandelt.

BASIC-BEFEHLE

In Kapitel 2 und 3 haben wir einige Befehle beschrieben, die Sie über das Tastenfeld eingeben können, um die Arbeit des CBM-Computers zu bestimmen. RUN ist ein solcher Befehl. Alle Befehle können in BASIC-Anweisungen ausgeführt werden.

Sie werden kaum Befehle außerhalb von BASIC-Anweisungen ausführen, wenn Sie mit der Programmierung beginnen.

Beim Schreiben sehr langer Programme kann Ihnen der Speicherplatz ausgehen. Man zerlegt dann ein Programm in eine Anzahl kleinere Module und läßt jeden von ihnen getrennt ablaufen. Ein kurzes Überwachungs-Programm verharrt im Speicher des Computers, um die Module nacheinander zu laden, Zwischenergebnisse in externe Speicher zu bringen und weitere Kontrollfunktionen im Computer auszuüben. Ein solches Überwachungs-Programm macht ausgiebig von Befehlen Gebrauch und ist in Kapitel 6 beschrieben.

RESERVIERTE WORTE

Alle Zeichenkombinationen, mit denen BASIC-Anweisungen beschrieben werden und alle Funktionen in BASIC stellen "reservierte Worte" dar. Tabelle 4-4 gibt eine Zusammenstellung aller reservierten Worte in CBM BASIC.

Tabelle 4-4 . Reservierte Worte für Befehle in CBM BASIC

Wort	Abkürzung		Wort	Abkürzung		Wort	Abkürzung		Wort	Abkürzung	
	Modus Groß-/ Kleinschrift	Modus Großschrift/ Graphik		Modus Groß-/ Kleinschrift	Modus Großschrift/ Graphik		Modus Groß-/ Kleinschrift	Modus Großschrift/ Graphik		Modus Groß-/ Kleinschrift	Modus Großschrift/ Graphik
ABS	aB	A|	DS$*	ds$	DS$	NEW	ne	NEW	SCRATCH*	sC	S-
AND	aN	A/	DSAVE*	dS	D♥	NEXT	nE	N¯	SGN	sG	S|
APPEND*	aP	A┐	END	eN	E/	NOT	nO	N┌	SIN	sI	S\
ASC	aS	A♥	EXP	eX	E♣	ON	on	ON	SPC(	sP	S┐
ATN	aT	A|	FN	fn	FN	OPEN	oP	O┐	SQR	sQ	S●
BACKUP*	bA	B♠	FOR	fO	F┌	OR	or	OR	ST	st	ST
CHR$	cH	C	FROM	fR	F_	PEEK	pE	P¯	STATUS	status	STATUS
CLOSE	cl0	CL┌	GET	gE	G¯	POKE	pO	P┌	STEP	stE	ST¯
CLR	cL	CL	GET #	ge+#	GET#	POS	pos	POS	STOP	sT	S|
CMD	cM	C\	GOTO	gO	G┌	PRINT	?	?	STR$	str#	STR#
COLLECT*	coL	COL	GOSUB	goS	GO♥	PRINT #	pR	P_	SYS	sY	S	
CONCAT*	conC	CON¯	HEADER*	hE	H¯	READ	rE	R¯	TAB(	tA	T♠
CONT	cO	C┌	IF	i+	IF	READ #	read#	READ#	TAN	tan	TAN
COPY*	coP	CO┐	INPUT	input	INPUT	RECORD*	reC	RE¯	THEN	tH	T	
COS	cos	COS	INPUT #	iN	I/	REM	rem	REM	TI	ti	TI
DATA	dA	D♠	INT	int	INT	RENAME*	reN	RE/	TIME	time	TIME
DCLOSE*	dC	D¯	LEFT$	leF	LE_	RESTORE	reS	RE♥	TI$	ti$	TI#
DEF	dE	D┌	LEN	len	LEN	RETURN	reT	RE|	TO	to	TO
DIM	dI	D\	LET	lE	L¯	RIGHT$	rI	R\	US	uS	U♥
DIRECTORY*	diR	DI_	LIST	lI	L\	RND	rN	R/	VAL	vA	V♠
DLOAD*	dL	DL	LOAD	lO	L┌	RUN	rU	R\	VERIFY	vE	V¯
DOPEN*	dO	D┌	LOG	log	LOG	SAVE	sA	S♠	WAIT	wA	W♠
DS*	ds	DS	MID$	mI	M\						

* nur in BASIC 4.0 und BASIC >4.0

Bei Programmausführung durchsucht der CBM-Computer jede BASIC-Anweisung nach dem Vorhandensein reservierter Worte. Lediglich Zeichenketten zwischen Anführungszeichen sind hiervon ausgenommen. Schwierigkeiten entstehen, wenn ein reserviertes Wort irgendwo im Namen einer Variablen auftritt. Der CBM-Computer ist nicht schlau genug, eine Variable an ihrer Stellung in einer BASIC-Anweisung zu erkennen. Sie müssen daher sehr sorgfältig darauf achten, reservierte Worte nicht in Variablennamen geraten zu lassen; das gilt besonders für kurze reservierte Worte, die leicht in einem Namen auftreten können.

Einige der reservierten Worte in Tabelle 4-4 sind mit einem Stern gekennzeichnet. Diese Worte sind nur in CBM BASIC, Version 4.0 und höher reserviert. Sie sollten sie aber auch in anderen Versionen von CBM BASIC nicht verwenden, um die Verträglichkeit des Programms mit anderen CBM-Computern zu erhalten.

ABGEKÜRZTE BEFEHLSWORTE

Schon am Anfang dieses Buches sahen Sie, daß die Anweisung PRINT am Tastenfeld auch durch ein Fragezeichen, ?, ersetzt werden kann. Dieses Zeichen wird im Computer automatisch durch das Wort PRINT ersetzt.

Die meisten Befehle, Anweisungen und Funktionen in BASIC können mit den ersten beiden Zeichen ihres Namens eingegeben werden, wenn das zweite Zeichen mit gedrückter Umschalttaste (SHIFT) eingegeben wird. Ist das Tastenfeld auf den Modus Großschrift/Graphik eingestellt, dann erscheint das zweite Zeichen als graphisches Symbol. Im Modus Groß/Kleinschrift ist das zweite Zeichen ein Großbuchstabe.

Tabelle 4-4 enthält eine Zusammenstellung aller Abkürzungen.

Es folgt ein Programmbeispiel, bei dessen Eingabe Abkürzungen verwendet wurden. Bei einer anschließenden Auflistung des Programms, die unten gezeigt ist, werden alle Abkürzungen durch das volle Wort ersetzt:

```
pO 59468,14                    Abkürzung für POKE (nach Ausführung)
10 lE a=10
20 b=a aN 14+eX(2)
30 dI c(5)
40 fO i=0 to 5
50 rE c(i)
60 nE
70 dA 1,6,2,4,10,5,16
80 re8
90 eN
lI                             Abkürzung für LIST
10 let a=10
20 b=a and 14+exp(2)
30 dim c(5)
40 for i=0 to 5
50 read c(i)
60 next
70 data 1,6,2,4,10,5,16
80 restore
90 end
pO 59468,12                    Abkürzung für POKE (vor Ausführung)
```

Nach Eingabe von RETURN am Ende der letzten Zeile (POKE-Anweisung) ist das Tastenfeld im Modus Großschrift/Graphik. Abkürzungen erscheinen dann mit graphischen Symbolen und die Auflistung in Großbuchstaben.

Beachten Sie folgendes: Bei den Funktionen SPC und TAB wird beim Übergang von der Abkürzung zum vollen Wort eine weitere linke Klammer hinzugefügt. Bei Eingabe der Abkürzung hat man hierauf zu achten:

```
10 ?sP(5)
```

Eine Auflistung macht hieraus:

```
10 print spc((5)
```

Syntax-Fehler: 2 Klammern

Die richtige Abkürzung muß folgendermaßen lauten:

```
10 ?sP5)
```

Diese Klammer-Regel gilt nur für die Funktionen SPC und TAB und ist eine Inkonsequenz in der Syntax. Bei allen anderen Funktionen geben Sie, wie erwartet, beide Klammern ein:

```
10 ?rN(1)
```

BASIC-ANWEISUNGEN

Anweisungen beschreiben dem Computer auszuführende Operationen und werden mit Befehlen formuliert, die in Tabelle 4-4, "Reservierte Worte", zusammengestellt sind.

Kapitel 8 enthält eine vollständige Beschreibung von Bedeutung und Anwendung aller Befehle in CBM BASIC aus Tabelle 4-4. Dieses Kapitel gibt eine Einführung

in die Grundzüge der Programmerstellung und besonders der Anwendung von Anweisungen. Die verwendeten Befehle werden nicht ausführlich beschrieben. Lesen Sie hierfür in Kapitel 8 nach, wenn Sie den Gebrauch eines der Befehle nicht verstehen.

KOMMENTARE: REM

Wir beginnen mit der Beschreibung einer BASIC-Anweisung, die als einzige vom Computer ignoriert wird: Der Kommentar, eingeleitet durch REM. Sind die ersten drei Zeichen einer BASIC-Anweisung REM, dann ignoriert der Computer diese Anweisung völlig. Warum also solch eine Anweisung verwenden? Die Antwort ist, daß Ihr Programm mit Kommentaren leichter zu lesen ist.

Bei einem Programm mit fünf oder zehn Anweisungen werden Sie wenig Mühe haben, sich an den Inhalt des Programms zu erinnern – es sei denn, Sie haben es für Monate nicht benutzt. Bei Programmen mit 100 oder noch mehr Anweisungen vergessen Sie sicherlich bald wichtige Einzelheiten über die verwendeten Anweisungen. Nach Dutzenden von Programmen, die Sie geschrieben haben, können Sie sich wahrscheinlich an keine Einzelheiten in den Programmen erinnern. Die Lösung sind Kommentare im Programm, um die Bedeutung wichtiger Programmabschnitte oder Anweisungen zu erläutern.

Gute Programmierer machen reichlich Gebrauch von Kommentaren. Zur Gewöhnung daran werden wir in den Programmbeispielen dieses Kapitels viele Kommentare verwenden.

Kommentare sehen aus wie Programmzeilen, d.h. sie beginnen mit einer Zeilennummer:

```
.
.
50 REM      FLAGGE F SETZEN:
60 IF X<Y AND B>40 THEN F=0
.
```

ZUWEISUNGEN (I): LET

Mit diesen Anweisungen können Sie einer Variablen Zahlen und Zeichenketten zuweisen. Zuweisungen treten sehr häufig in BASIC-Programmen auf. Hier einige Beispiele:

```
100 REM WERTZUWEISUNG ZUR VARIABLEN X
110 LET X=3.24
```

Der Gleitkomma-Variablen X wird in Zeile 110 der Wert 3.24 zugewiesen.

```
150 X=3.24
```

Die Verwendung von LET in der Wertzuweisung kann unterbleiben.

```
200 A$="SPACE SHUTTLE"
```

Der Zeichenketten-Variablen A$ wird in Zeile 200 der Text SPACE SHUTTLE zugewiesen

Mit folgenden Anweisungen können wir einer Tabelle MP$ Eintragungen MP$ (I) zuweisen:

```
200 REM  MP$(I) = VARIABLE FUER MILCHPRODUKTE
210 MP$(0)= "MILCH"
220 MP$(1)= "SAHNE"
230 MP$(2)= "HUETTENKAESE"
```

Da in einer Programmzeile mehrere Anweisungen stehen dürfen, können Zuweisungen zu MP$ auch wie folgt aussehen:

```
200 REM  MP$(I) = VARIABLE FUER MILCHPRODUKTE
210 MP$(0)="MILCH":MP$(1)="SAHNE":MP$(2)="HUETTENKAESE"
```

Vergessen Sie nicht den Doppelpunkt zwischen benachbarten Anweisungen.

In einer Zuweisung kann jede der früher besprochenen arithmetischen oder vergleichenden Operatoren stehen. Hier ein Beispiel:

```
100 REM  BEISPIEL EINER UNUEBERSICHTLICHEN WERTZUWEISUNG V
110 V=3.24+7.96/8.5
```

Diese Anweisung weist der Gleitkomma-Variablen V den Zahlenwert 4.17647059 zu und ist mit folgenden drei Anweisungen gleichwertig:

```
100 REM WERTZUWEISUNGEN X,Y VOR WERTZUWEISUNG V
110 X=7.96
120 Y=8.5
130 V=3.24+X/Y
```

die auch in einer Programmzeile geschrieben werden können:

```
100 X=7.96:Y=8.5:V=3.24+X/Y
```

Schließlich Zuweisungen, in denen logische Operatoren vorkommen:

```
100 A%=43 AND 137
110 B%=43 OR 137
```

Die Zuweisung von Zeichenketten kann auch in Verbindung mit einer Verkettung erfolgen, wie dieses Beispiel zeigt:

```
100 REM  MP$(2) WIRD DER WERT "SAURE SAHNE" ZUGEWIESEN
200 V$= "SAURE"
300 W$= "SAHNE"
400 MP$(2)=V$+" "+W$
```

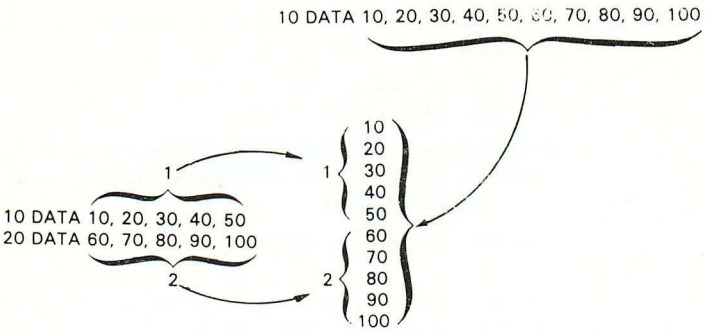
ZUWEISUNGEN (II): DATA . . . READ

Wenn mehrere Variable in einem Programm Zuweisungen zu Daten benötigen, dann sollten die Anweisungen DATA und READ benutzt werden. Betrachten Sie folgendes Beispiel:

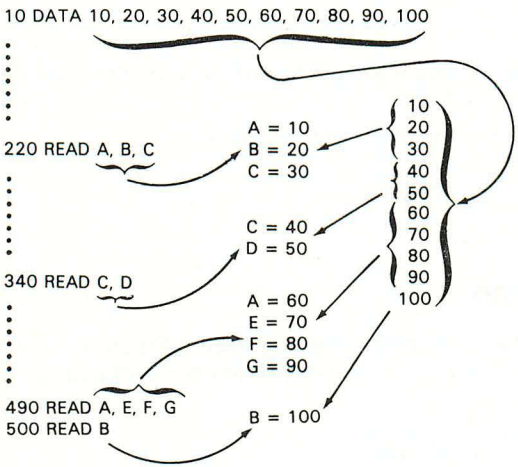
```
10 REM  INITIIEREN ALLER PROGRAMM-VARIABLEN
20 REM  DURCH WERTZUWEISUNGEN MIT DATA/READ:
30 DATA 10,20,-4,16E6
40 READ A,B,C,D
```

Die Anweisung DATA in Zeile 30 hält vier Zahlenwerte gespeichert, die mit der Anweisung READ in Zeile 40 den Gleitkomma-Variablen A, B, C, D zugewiesen werden. Nach Ausführung der Anweisungen DATA, READ sind folgende Zuweisungen erfolgt: A = 10, B = 20, C = –4, D = 16×10^6

Haben Sie eine oder mehrere DATA-Anweisungen in Ihrem Programm, dann kann man sich vorstellen, daß sie eine Spalte von Zahlen bilden. Eine DATA-Anweisung mit 10 Zahlen würde die gleiche Zahlenspalte bilden wie zwei DATA-Anweisungen mit je fünf Zahlen, wie folgendes Bild zeigt:

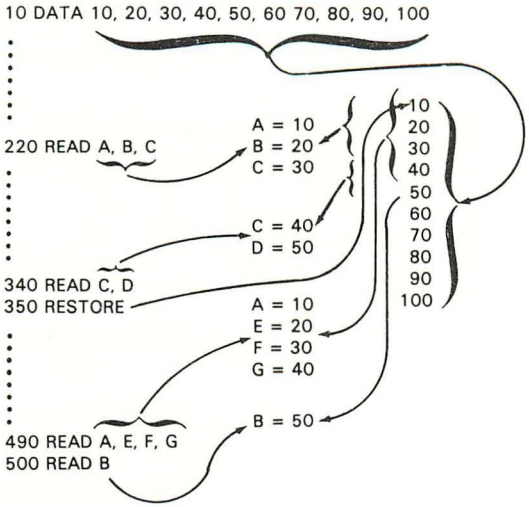


Die erste READ-Anweisung im Programm beginnt dann das Lesen der Daten in DATA mit der ersten Zahl. Weitere READ-Anweisungen im Programm setzen das Lesen fort, wie folgendes Schema veranschaulicht:



ZUWEISUNGEN (III): DATA . . . READ . . . RESTORE . . .

Mit der Anweisung RESTORE kann READ auf den Anfang der Daten in DATA zurückgesetzt werden. Hier ein Beispiel für fortlaufendes Lesen von Daten (READ A, B, C und READ C, D) und wiederholtes Lesen (RESTORE in Zeile 350; READ A, E, F, G):



ERKLÄRUNG VON TABELLEN: DIM

Im einfachsten Fall nimmt CBM BASIC für den Index einer Tabelle die Werte 0 bis 10 an. Dies ergibt eine Tabelle für elf Einträge. Wünschen Sie mehr oder weniger als 11 Einträge, dann muß diese Änderung dem Computer in einer Dimensions-Anweisung mitgeteilt werden. Diese Anweisung ist auch immer dann erforderlich, wenn eine Tabelle mit zwei oder mehr Indizes angelegt werden soll. Das folgende Beispiel benutzt Variablennamen aus dem Abschnitt über Tabellen:

```
DIM FG$(5),OG$(3),MP$(2)
```

Die 2-dimensionale Einkaufsliste EL$ müßte folgendermaßen erklärt werden:

```
DIM EL$(3,5)
```

Eine Dimensions-Anweisung kann für eine beliebige Zahl von Variablen verwendet werden, solange die entstehende Programmzeile nicht mehr als 80 Zeichen enthält.

Die Zahlen in den Variablennamen einer DIM-Anweisung bezeichnen den größten Wert, den ein Index an dieser Stelle annehmen kann. Aber erinnern Sie sich: Indizes beginnen bei Null. DIM MP$ (5) bedeutet daher, daß die Tabelle MP$ sechs Einträge hat: 0, 1, 2, 3, 4, 5. DIM EL$ (3, 5) erklärt eine Tabelle für 24 Einträge.

Sobald eine Dimensions-Anweisung den Wertebereich der Indizes einer Tabelle erklärt hat, müssen Sie sich im Programm an diese Erklärung halten. Beim Überschreiten eines in DIM erklärten Maximalwerts erfolgt eine Fehlermeldung.

PROGRAMMVERZWEIGUNGEN

Anweisungen in einem BASIC-Programm werden in der Regel in aufsteigender Ordnung der Zeilennummern ausgeführt. Dieser Vorgang wurde weiter oben in

diesem Kapitel beschrieben. Anweisungen zur Verzweigung ändern diese Ordnung der Programmausführung.

VERZWEIGUNGEN I: UNBEDINGTES GOTO

Die unbedingte GOTO-Anweisung ist die einfachste Verzweigungsanweisung; sie ermöglicht, die nächste auszuführende Anweisung anzugeben. Betrachten Sie das folgende Beispiel:

```
20 A=4.37
30 GOTO 100
40
50
60
70
80
90
100
110
.
.
```

Die Anweisung in Zeile 20 weist der Gleitkommavariablen A einen Wert zu. Die nächste Anweisung ist ein GOTO; sie gibt an, daß die Programmausführung mit Zeile 100 fortsetzen muß. In diesem Programmteil werden daher Programmzeilen in folgender Reihenfolge ausgeführt: Zeile 20, Zeile 30, Zeile 100.

Natürlich müssen später im Programm Anweisungen auftreten, die zurück auf Zeile 40 verzweigen, da sonst die Zeilen 40 bis 90 nie ausgeführt werden.

Die Verzweigung kann zu jeder Zeilennummer erfolgen, auch zu einer Zeile mit einem Kommentar. Der Computer ignoriert jedoch die Bemerkung, so daß die Wirkung die gleiche ist, als ob zur nächsten Zeile verzweigt worden wäre:

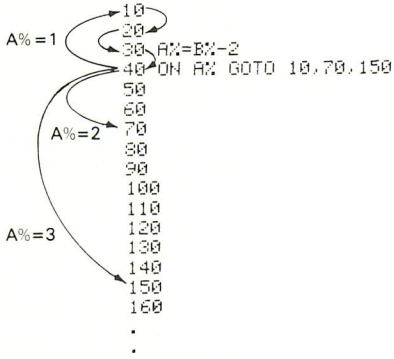
```
20  A=4.37
30  GOTO 70
40
50
60
70  REM  DIESE ZEILE ENTHAELT EINE BEMERKUNG,SONST NICHTS
80
90
.
.
```

Sie könnten genausogut auf die Zeile nach dem Kommentar verzweigt haben.

```
20  A=4.37
30  GOTO 80
40
50
60
70  REM  DIESE ZEILE ENTHAELT EINE BEMERKUNG,SONST NICHTS
80
90
.
.
```

VERZWEIGUNGEN II: BEDINGTES GOTO

Eine andere Form der GOTO-Anweisung ermöglicht die Verzweigung auf eine unter zwei oder mehr verschiedenen Zeilennummern, abhängig von dem augenblicklichen Wert einer Variablen, wie das folgende Beispiel veranschaulicht:



In Zeile 40 steht ein bedingter GOTO-Befehl. Die Bedingungen für den Sprung werden in Zeile 30 festgelegt: Für A% = 1 erfolgt der Sprung nach Zeile 10, für A% = 2 erfolgt der Sprung nach Zeile 70, für A% = 3 erfolgt der Sprung nach Zeile 150. Nimmt A% irgendeinen anderen Zahlenwert als 1, 2, 3 an, dann wird der GOTO-Befehl übersprungen (und das Programm mit Zeile 50 fortgesetzt). Zur Ausführung der Sprünge muß bei der Wertzuweisung A% = B% – 2 in Zeile 30 die Variable B% einen der Werte 3, 4 oder 5 annehmen.

Hier ein Versuchsprogramm mit bedingten und unbedingten GOTO-Anweisungen:

```
READY.

 10 BX=3
 20 PRINT BX;
 30 AX=BX-2
 40 ON AX GOTO 50,70,10
 50 BX=4
 60 GOTO 20
 70 BX=5
 80 GOTO 20
```

Zum Ablauf dieses Programms geben Sie am Tastenfeld RUN ein. Auf dem Bildschirm erscheint die fortlaufende Zahlenfolge, 345345345 . . .

PROGRAMMSCHLEIFEN

SCHLEIFEN I: FOR . . . . NEXT

Bedingte und unbedingte GOTO-Anweisungen ermöglichen, die Reihenfolge der Ausführung von Anweisungen nach den Erfordernissen der Programmlogik festzulegen. Nehmen Sie aber den Fall an, daß Sie eine Anweisung oder eine Gruppe von Anweisungen wiederholt ausführen wollen: In Tabelle A (I) sollen 100 Einträge erfol-

gen. Im Programm müßten 100 Zuweisungen der Form A (0) = ..., A (1) = ... usw. stehen. Weitaus einfacher ist es, diese 100 Zuweisungen mit einer FOR-NEXT-Anweisung auszuführen:

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
40 NEXT I
```

Die Anweisungen zwischen FOR und NEXT werden wiederholt ausgeführt. In unserem Beispiel wird die Anweisung A (I) = I wiederholt ausgeführt, bis 100 Eintragungen in die Tabelle erfolgt sind.

Als Eintragungen müßten in der Tabelle die Zahlen 0 bis 99 stehen. Mit dem folgenden Programm werden wir das Arbeiten der FOR-NEXT-Schleife prüfen, indem wir die Eintragungen in die Tabelle A (I) auf dem Bildschirm ausgeben:

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
40 PRINT A(I);
50 NEXT I
60 REM  DIE GOTO-ANWEISUNG IN ZEILE 80 VERZWEIGT AUF
70 REM  SICH SELBST,D.H. BILDET EINE WARTESCHLEIFE :
80 GOTO 80
```

Geben Sie zum Start des Programms RUN ein. Auf dem Bildschirm werden die Zahlen 0 bis 99 dargestellt. Drücken Sie die Taste STOP zum Beenden des Programms.

Anweisungen zwischen FOR und NEXT werden so oft ausgeführt, bis die Schleifenvariable I in Schritten von 1 (STEP 1) alle Zahlenwerte von 0 bis 99 durchlaufen hat. Die Schleifenvariable I erscheint auch in der Zuweisung A (I) = I. Beim ersten Schleifenzyklus ist I = 0, sodaß die erste Zuweisung wie folgt aussieht:

```
30 A(0)=0
```

Die Schrittweite STEP kann jeden ganzzahligen Wert haben. Ein Beispiel: Ändern Sie Zeile 20 in STEP 5 und lassen das Programm ablaufen. Die Schleife wird jetzt 19 x durchlaufen: 19 Erhöhungen von I um jeweils fünf ergibt 95; der zwanzigste Schleifendurchlauf findet nicht statt. Beginnend mit A (0) = 0 ergeben sich 20 Zuweisungen. Bei STEP 5 könnten wir 100 Zuweisungen wie zuvor dadurch erzeugen, daß wir den Maximalwert der Schleifenvariablen I von 99 auf 500 erhöhen. Gleichzeitig muß auch die Dimensions-Anweisung in DIM A (500) verändert werden.

Die Schleifenschritte STEP müssen nicht unbedingt positiv sein. Bei negativen Schleifenschritten muß der Anfangswert von I größer sein als der Endwert. Ein Beispiel: Für Zuweisungen der Zahlen 99 bis 0 zur Tabellenvariablen A (I) in Schritten von −1, müßten wir Zeile 20 wie folgt ändern:

```
10 DIM A(99)
20 FOR I=99 TO 0  STEP -1
30 A(I)=I
40 PRINT A(I);
50 NEXT I
80 GOTO 80
```

Anfangs- und Endwert der Schleifenvariablen I und die Schrittweite werden als Ganzzahlen ausgewertet; weitere Beschränkungen bestehen nicht für diese Zahlen. Sie können diese drei Zahlen auch als Gleitkommavariable oder Ausdrücke angeben. Gleitkomma-Ergebnisse dieser Ausdrücke werden gerundet und als Ganzzahlen dargestellt. Da die Regeln der Rundung, wie am Kapitelanfang beschrieben, mit Problemen

verbunden sind, wird Ihnen dringend empfohlen, Anfangswert, Endwert und Schrittweite der Schleifenvariablen als Ganzzahlen anzugeben. Benutzen Sie nicht stattdessen Ausdrücke, die das Programm unnötig komplizieren. Wenn Sie einen dieser Schleifenwerte berechnen müssen, geschieht dies einfacher und schneller in einer eigenen Anweisung.

Bei einer Schrittweite von 1 erübrigt sich die Angabe von STEP. Bei fehlender STEP-Angabe nimmt CBM BASIC automatisch eine Schrittweite von 1 an. Zeile 20 könnte daher auch wie folgt lauten:

```
10 DIM A(99)
15 REM  BEI FEHLENDER ANGABE IST
16 REM  DIE SCHRITTWEITE 1 (STEP=1):
20 FOR I=0 TO 99
30 A(I)=I
40 PRINT A(I);
50 NEXT I
80 GOTO 80
```

Die Angabe der Schleifenvariablen in NEXT kann ebenfalls entfallen. Bei verschachtelten FOR-NEXT-Schleifen erleichtert jedoch die Angabe der Schleifenvariablen das Lesen des Programms.

SCHLEIFEN II: VERSCHACHTELTES FOR-NEXT

Die FOR-NEXT-Struktur wird "Programmschleife" genannt, da die Ausführung der Anweisung zwischen FOR-NEXT-FOR zirkuliert. Diese Schleifenstruktur tritt häufig auf; fast jedes BASIC-Programm, das Sie schreiben, enthält eine oder mehrere derartige Schleifen. Da die Zahl der Anweisungen zwischen FOR und NEXT beliebig groß sein kann, können auch weitere FOR-NEXT-Schleifen in diesen Anweisungen auftreten. Man nennt derartige FOR-NEXT-Schleifen dann verschachtelt. Zur Veranschaulichung einer einfachen Verschachtelung das folgende Beispiel:

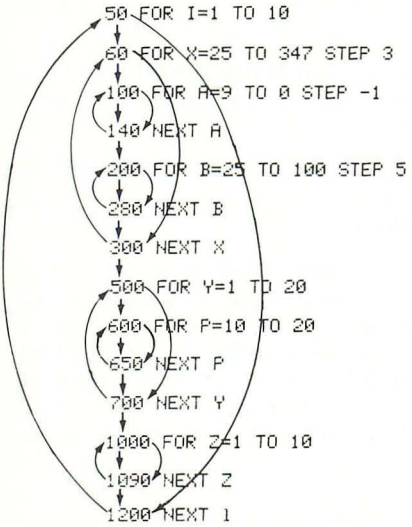
```
10 DIM A(99)
20 FOR I=0 TO 99
30 A(I)=I
40 REM  DARSTELLUNG ALLER BISHERIGEN WERTE VON A(I):
50 FOR J=0 TO I
60 PRINT A(J)
70 NEXT J
80 NEXT I
90 GOTO 90
```

Das folgende Beispiel zeigt mehrfache Verschachtelungen von FOR-NEXT-Schleifen:

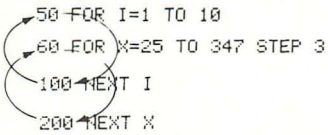
```
50 FOR I=1 TO 10
60 FOR X=25 TO 347 STEP 3
.
100 FOR A=9 TO 0 STEP -1
.
140 NEXT A
200 FOR B=25 TO 100 STEP 5
.
280 NEXT B
300 NEXT X
.
500 FOR Y=1 TO 20 STEP 2
.
600 FOR P=10 TO 20
```

```
.
650 NEXT P
700 NEXT Y
.
1000 FOR Z=1 TO 10
.
1090 NEXT Z
1200 NEXT I
```

Die äußerste Schleife benutzt den Schleifenindex I und enthält drei verschachtelte Schleifen mit den Indizes X, Y, C. Die äußere Schleife enthält zusätzlich zwei weitere Schleifen mit den Indizes A und B. Die zweite Schleife enthält eine verschachtelte Schleife mit dem Schleifenindex P. Die dritte Schleife enthält keine Verschachtelung. Bei verschachtelten Schleifen muß die Schleifenvariable jeder Schleife einen anderen Namen tragen. Hier eine weitere Veranschaulichung des zuletzt beschriebenen Beispiels:



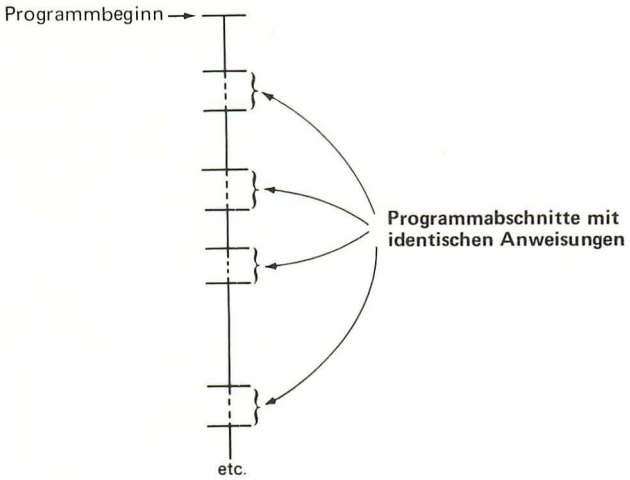
Programmschleifen sind sehr anschaulich und leicht zu benutzen. Es gibt nur einen häufigen Fehler, den Sie vermeiden müssen: Schließen Sie eine äußere Schleife nicht eher mit NEXT ab, ehe die innere Schleife abgeschlossen ist. Die folgende Schleifenstruktur ist unzulässig:



In jedem Programm müssen soviele FOR-Anweisungen auftreten, wie es NEXT-Anweisungen gibt. Ihr Computer meldet einen Syntax-Fehler, wenn mehr oder weniger NEXT-Anweisungen als FOR-Anweisungen auftreten.

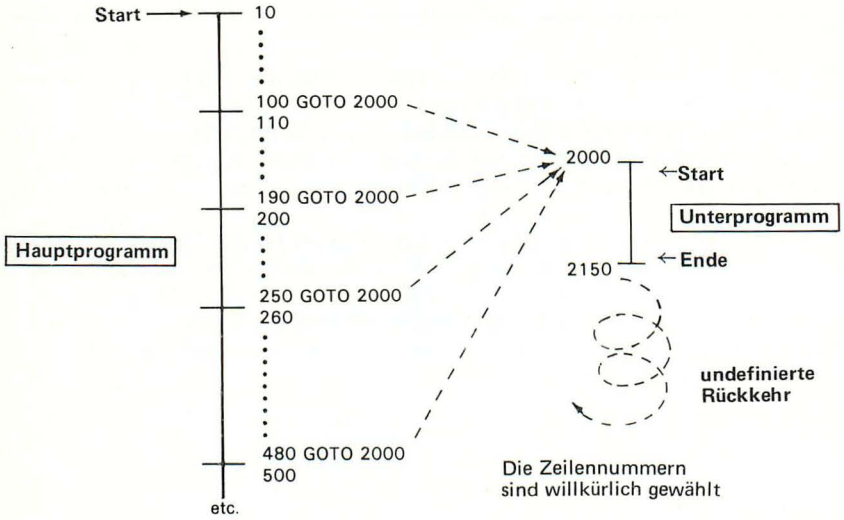
Sobald Sie anfangen, längere Programme zu schreiben, werden Sie schnell sich häufig wiederholende Anweisungen finden. Denken Sie an eine Tabelle, in die mehrmals in Ihrem Programm Einträge erforderlich sind: Würden Sie einfach die drei Anweisungen der FOR-NEXT-Schleife wiederholen, mit denen wir oben die Zuweisungen zu A (I) ausgeführt haben? In diesem Fall sind es nur drei Anweisungen, und Sie könnten es tun.

Nehmen Sie aber den Fall an, daß mit der Zuweisung zur Tabellenvariablen A (I) auch noch 10 oder mehr Anweisungen verbunden sind, um die Tabelleneintragungen zu bearbeiten. Diese mehr als 10 Anweisungen jedesmal im Programm zu wiederholen, kostet nicht nur Zeit bei der Programmerstellung, sondern, viel wichtiger, es wird sehr viel Speicherplatz im Computer vergeudet. Das Geschilderte kann man wie folgt veranschaulichen:



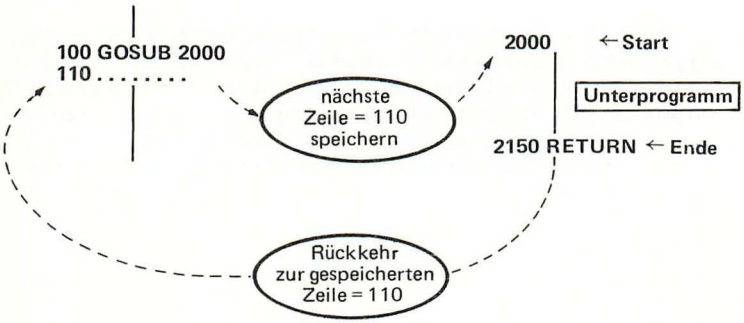
Könnte man nicht die wiederkehrenden Programmabschnitte herausnehmen und zu ihnen verzweigen, wenn im Programm erforderlich? Genau das wird geschehen; diese wiederkehrenden Programmabschnitte nennt man dann "Unterprogramm".

Ein Problem gibt es noch. Von Ihrem Programm zu einem Unterprogramm zu verzweigen, ist einfach; das Unterprogramm hat ja auch Zeilennummern; was soll aber am Ende des Unterprogramms geschehen? Man könnte mit einer GOTO-Anweisung zum Unterprogramm verzweigen:



SPRUNG IN UNTERPROGRAMME I: GOSUB

Wohin soll am Ende des Unterprogramms die Rückkehr erfolgen? Wenn zwei GOTO-Anweisungen zum Unterprogramm verzweigen, dann gibt es am Ende des Unterprogramms auch zwei Stellen, zu denen die Rückkehr erfolgen muß. Die Antwort für dieses Problem sind besondere Anweisungen: Statt einer Verzweigung zum Unterprogramm mit GOTO, verwenden Sie die Anweisung GOSUB. Diese Anweisung verzweigt wie eine GOTO-Anweisung, zusätzlich aber speichert sie die nächste Zeilennummer im Hauptprogramm, wohin die Rückkehr aus dem Unterprogramm erfolgen soll:



Beenden Sie das Unterprogramm mit einer RETURN-Anweisung. Diese Anweisung löst den Rücksprung ins Hauptprogramm aus, und zwar zu der in GOSUB gespeicherten Zeilennummer. Unsere weiter oben benutzte FOR-NEXT-Schleife für Zuweisungen zu A (I) würde als Unterprogamm wie folgt aussehen:

```
10 REM      ** HAUPTPROGRAMM **
20 REM DIE DIMENSIONS-ANWEISUNG FUER VARIABLE IM
30 REM UNTERPROGRAMM SOLLTE IM HAUPTPROGRAMM STEHEN:
60 DIM A(99)
70 GOSUB 2000
80 REM SICHTBARMACHEN DES RUECKSPRUNGS:
90 PRINT "RUECKSPRUNG IST ERFOLGT"
100 GOTO 100
2000 REM     ** UNTERPROGRAMM  **
2010 FOR I=1 TO 99
2020 A(I)=I
2030 PRINT A(I);
2040 NEXT I
2050 RETURN
```

VERSCHACHTELTE UNTERPROGRAMME

Unterprogramme können auch verschachtelt sein. Ein Unterprogramm kann selbst in ein anderes Unterprogramm verzweigen, das seinerseits in ein drittes Unterprogramm verzweigen kann, usw. Es gibt nichts Neues zu sagen zu verschachtelten Unterprogrammen. Springen Sie einfach zum gewünschten Unterprogramm mit GOSUB und beenden Sie das Unterprogramm jeweils mit RETURN. In CBM BASIC wird die jeweils richtige Zeilennummer für die Rückkehr gespeichert. Das folgende Beispiel gibt eine Vorstellung von verschachtelten Unterprogrammen:

```
 10 REM       ** HAUPTPROGRAMM **
 20 REM DIE DIMENSIONS-ANWEISUNG FUER VARIABLE IM
 30 REM UNTERPROGRAMM SOLLTE IM HAUPTPROGRAMM STEHEN:
 60 DIM A(99)
 70 GOSUB 2000
 80 REM SICHTBARMACHEN DES RUECKSPRUNGS:
 90 PRINT "RUECKSPRUNG IST ERFOLGT"
 100 GOTO 100
 2000 REM     ** UNTERPROGRAMM  **
 2010 FOR I=1 TO 99
 2020 A(I)=I
 2030 GOSUB 3000
 2040 NEXT I
 2050 RETURN
3000 REM  ** VERSCHACHTELTES UNTERPROGRAMM **
3010 PRINT A(I);
3020 RETURN
```

In diesem Programm wird die Anweisung PRINT A (I) aus dem Zuweisungs-Unterprogramm in ein verschachteltes Programm zur Darstellung von A (I) verlagert. Sonst ist nichts verändert.

SPRUNG IN UNTERPROGRAMME II: BEDINGTES GOSUB

Die Anwendung von GOTO und GOSUB ist sehr ähnlich. Der einzige Unterschied zwischen beiden ist, daß GOSUB sich die nächste Zeilennummer merkt. Es ist daher nicht überraschend, daß es zum bedingten GOTO-Sprung eine analoge bedingte GOSUB-Anweisung gibt. Die bedingte GOSUB-Anweisung ermöglicht, auf eins von zwei oder mehr Unterprogrammen zu verzweigen, in Abhängigkeit vom Wert einer Variablen:

```
90
100 ON A GOSUB 1000,500,5000,2300
110
```

Die Anweisung in Zeile 100 besagt: Wenn A = 1, dann erfolgt der Sprung zum Unterprogramm mit der ersten Zeilennummer 1000, wenn A = 2, dann erfolgt der Sprung zum Unterprogramm mit der ersten Zeilennummer 500 usw. Wenn A einen anderen Wert als 1, 2, 3, 4 annimmt, endet der Programmlauf und eine Fehlermeldung erscheint. Auch das bedingte GOSUB speichert die folgende Zeilennummer, im Beispiel 110.
In jedem der Unterprogramme sorgt die RETURN-Anweisung des angesprungenen Unterprogramms für die Rückkehr zur gespeicherten Zeilennummer des Hauptprogramms, in unserem Beispiel Zeile 110.

Auch mit bedingten GOSUB-Anweisungen sind verschachtelte Unterprogramme möglich.

ENTSCHEIDUNGEN: IF . . . . THEN

Die am Kapitelanfang beschriebenen arithmetischen und vergleichenden Operatoren werden häufig im Zusammenhang mit IF-THEN-Anweisungen benutzt. Hierdurch ist ein BASIC-Programm in der Lage, Entscheidungen zu treffen. Nach IF geben Sie einen Ausdruck ein: Ist der Ausdruck "WAHR", dann werden die nach THEN stehenden Anweisungen ausgeführt. Ist der Ausdruck jedoch "FALSCH", dann werden Anweisungen nach THEN nicht ausgeführt. Hier drei Beispiele für die Anwendung von IF-THEN-Anweisungen:

```
10 IF A=B+5 THEN PRINT MSG1
40 IF CC$<"M" THEN IN=0
50 IF Q<14 AND M<>M1 GOTO 66
```

Das Wort THEN kann fortgelassen werden, wie das dritte Beispiel zeigt.

Die Anweisung in Zeile 10 macht die Ausführung der PRINT-Anweisung abhängig von der Bedingung, daß die Gleitkommavariable A um 5 größer ist als die Gleitkommavariable B. Sonst wird die PRINT-Anweisung nicht ausgeführt.

Die Anweisung in Zeile 40 setzt die Gleitkommavariable IN auf den Wert Null, wenn die Zeichenkette CC$ einen der Buchstaben A bis L des Alphabets darstellt.

Die Anweisung in Zeile 50 läßt das Programm auf Zeile 66 verzweigen, wenn die Gleitkommavariable Q kleiner als 14 ist und die Gleitkommavariablen M, M1 ungleich sind. Sind diese Bedingungen nicht erfüllt, dann fährt das Programm mit der Anweisung auf der nächsten Programmzeile fort.

Wenn Sie Schwierigkeiten mit Ausdrücken haben, die nach IF verwendet werden, dann holen Sie sich Rat von der Diskussion derartiger Ausdrücke am Anfang dieses Kapitels.

EINGABE- UND AUSGABE-ANWEISUNGEN

In BASIC gibt es eine Reihe von Anweisungen zur Übertragung von Daten zum und vom Computer. Diese Anweisungen werden zusammenfassend als Ein/Ausgabe-Anweisungen bezeichnet. Die einfachsten Ein/Ausgabe-Anweisungen kontrollieren die Dateneingabe vom Tastenfeld und die Datenausgabe zum Bildschirm. Im folgenden Paragraphen werden wir diese einfachen Ein/Ausgabe-Anweisungen bespre-

chen. Ein/Ausgabe-Anweisungen für den Datenverkehr zwischen Computer und Peripherie-Geräten wie Kassette, Diskette und Drucker, werden zusammen mit der Beschreibung dieser Geräte behandelt (s. Kap. 6).

Da wir schon die PRINT-Anweisung kennen, wollen wir diese zuerst besprechen.

DATENAUSGABE: PRINT

Diese Anweisung kann durch das Wort PRINT oder ein Fragezeichen (?) dargestellt werden.

Warum wird für den Datenverkehr mit dem Bildschirm das Wort PRINT benutzt? Die Antwort ist einfach: Am Anfang der 60er Jahre, als die Programmiersprache BASIC entwickelt wurde, waren Bildschirme sehr teuer und nicht allgemein verfügbar für Computer der mittleren und unteren Preisklasse. Eine typische Computer-Station bestand aus einem Tastenfeld und einem Drucker für die Eingabe und Ausgabe von Informationen.

Die PRINT-Anweisung dient zur Darstellung von Text oder Zahlen auf dem Bildschirm. Text muß in Anführungszeichen eingeschlossen sein. Die folgende Anweisung stellt beispielsweise das Wort "TEXT" dar:

```
10 PRINT "TEXT"
```

oder:

```
10 ?"TEXT"
```

Zur Darstellung von Zahlen setzen Sie einfach die Zahl oder den Variablen-Namen hinter die PRINT-Anweisung:

```
10 AX=10
20 ?5,AX
```

Die Anweisung in Zeile 20 bringt die Zahl 5 und danach die Zahl 10 in der gleichen Bildschirmzeile zur Darstellung.

Sie können auch beides, Text und Zahlen, mit der gleichen PRINT-Anweisung zur Darstellung bringen. Trennen Sie die Aufzählung der einzelnen Informationnen durch Kommas:

```
10 PRINT "EINS",1,"ZWEI",2,"DREI",3,"VIER",4,"FUENF",5
```

Wenn Sie Informationen in der Aufzählung einer PRINT-Anweisung durch Kommas trennen, wie das Beispiel oben zeigt, dann bringt der CBM-Computer jede Information nacheinander auf den Spalten 1, 11, 21, . . . 71 zur Darstellung. Machen Sie in direktem Dialog einen Versuch mit der angegebenen Anweisung. Wenn Sie bei der Darstellung keine Zwischenräume zwischen den Informationen wünschen, benutzen Sie als Trennungszeichen das Semikolon:

```
10 PRINT "EINS";1;"ZWEI";2;"DREI";3;"VIER";4;"FUENF";5
```

Geben Sie auch diese Anweisung in direktem Dialog ein, um die Wirkung eines Semikolons zu sehen.

Eine PRINT-Anweisung beendet automatisch ihre Bildschirmdarstellung mit einem Zeilensprung (RETURN), den Sie aber unterdrücken können, indem Sie die Liste der Variablen einer PRINT-Anweisung mit einem Komma oder Semikolon abschließen. Ein Komma bewirkt einen Abstand von 10 Zeichen bis zur Darstellung der nächsten PRINT-Anweisung. Machen Sie gleich einen Versuch mit dem folgenden Programm:

```
10 PRINT "EINS",1,"ZWEI",2,
20 PRINT "DREI",3,"VIER",4
30 GOTO 30
```

Geben Sie RUN zum Start und die Taste STOP zum Stop des Programms ein. Die Angaben in den PRINT-Anweisungen werden zunächst auf zwei Bildschirmzeilen dargestellt. Wenn Sie ein Komma an das Ende der Anweisung in Zeile 10 anfügen, erfolgt die Darstellung beider PRINT-Anweisungen in einer Zeile: Der Zeilensprung wurde unterdrückt.

Ersetzen Sie jetzt das letzte Komma in Zeile 10 durch ein Semikolon: Die Darstellung erfolgt auch auf einer Bildschirmzeile, aber ohne Abstand zwischen der Zahl 2 und dem Wort DREI. Durch Umwandlung weiterer Kommas in Semikolons können Sie auch die übrigen Zwischenräume zum Verschwinden bringen.

In unseren letzten Beispielen haben wir darzustellende Zahlen direkt in die PRINT-Anweisungen geschrieben. Diese Zahlen könnten auch durch Variable ausgedrückt werden, deren Inhalt dann mit PRINT dargestellt wird. Das folgende Programm verwendet die Variable A% (I) zur Darstellung der Zahlen 1 bis 5:

```
10 FOR I=1 TO 5
20 A%(I)=I
30 NEXT
40 PRINT "EINS";A%(1);"ZWEI";A%(2);"DREI";A%(3);
45 PRINT "VIER";A%(4);"FUENF";A%(5)
50 GOTO 50
```

Mit den Anweisungen DATA-READ könnten wir auch die Zeichenketten in PRINT durch eine Variable darstellen:

```
10 DATA "EINS","ZWEI","DREI","VIER","FUENF"
20 FOR I=1 TO 5
30 A%(I)=I
40 READ N$(I)
50 PRINT N$(I);A%(I);
60 NEXT I
70 GOTO 70
```

Dieses Programm ist nicht sehr gut geschrieben. Die Zahlen I müssen nicht erst über den Umweg A% (I) in die PRINT-Anweisung gelangen, und für das Lesen der Zeichenketten durch READ muß nicht eine Tabelle N$ (I) angelegt werden. Versuchen Sie, das Programm zu ändern.

BILDSCHIRMGESTALTUNG

Wir benutzen das Wort "Gestaltung" oder "Formatierung" bei der Beschreibung von Anweisungen, mit denen Informationen derart auf einem Bildschirm (oder Drucker) angeordnet werden können, daß sie leichter zu verstehen sind und für das Auge ein angenehmes Erscheinungsbild ergeben. Eine Formatierung wäre mit der PRINT-Anweisung sehr schwierig. Ein Beispiel: Die Position von Spaltenüberschriften müßte in einer PRINT-Anweisung durch Verwendung von Leerstellen festgelegt werden. Dieses Verfahren wäre mühsam und fehleranfällig, aber insbesondere geht viel Speicherplatz verloren, da jede Leerstelle im Computer einen Speicherplatz belegt. Zum Glück bietet CBM BASIC neben PRINT noch drei weitere Anweisungen zur Gestaltung von Schirmbildern (und Ausdrucken): SPC, TAB, POS.

SPALTENFUNKTION: SPC

Die Funktion SPC hat die gleiche Wirkung wie die Taste CURSOR→. Der Kursor springt um die in der Klammer angegebene Zahl von Zeichenpositionen nach rechts. SPC wird wie eine Variable in die Liste von PRINT-Anweisungen eingetragen. Ein Beispiel: Eine Spaltenüberschrift würde mit folgendem PRINT am Bildschirmanfang dargestellt werden:

```
10 PRINT "UEBERSCHRIFT"
```

Zur Darstellung der Überschrift in der Mitte eines 40spaltigen Bildschirms würden wir die Spaltenfunktion SPC wie folgt verwenden:

```
10 PRINT SPC(16);"UEBERSCHRIFT"
```

Beachten Sie das Semikolon nach SPC: Ein Komma nach SPC würde die Darstellung des Textes um 10 Spalten nach rechts verschieben.

Tritt SPC in der Parameterliste einer PRINT-Anweisung auf, dann wird einfach der nächste Text in einer PRINT-Anweisung um die von SPC angegebene Zahl von Spalten nach rechts verschoben dargestellt; die Syntax der PRINT-Anweisung bleibt hiervon unberührt.

TABULIERFUNKTION: TAB

Die Tabulierfunktion TAB dient wie bei Schreibmaschienen zum Anlegen von Tabellen. Denken Sie an Fälle, in denen Information spaltenweise dargestellt werden soll. Zunächst müssen Sie durch Angabe von Zeichenzahlen in einer Zeile die Spaltenpositionen festlegen, wie das Beispiel für einen 80spaltigen Bildschirm zeigt:

Festlegen der Spaltenposition

↓	↓	↓	↓
0	16	32	48
JONES, P. J	431-25-6277	1420.00	258.74
BURKE, P. L	447-71-7614	2025.00	467.64
ROBINSON, L. W	231-80-8421	2150.00	477.04
etc.	etc.	etc.	etc.

In diesem Beispiel beginnen die Spalten an den Zeichenpositionen 0, 16, 32, 48. Betrachten wir jetzt die erste Zeile der Spalteneintragungen; durch Abzählen der Zeichenpositionen zwischen den Spalten könnten wir eine PRINT-Anweisung ohne TAB wie folgt formulieren:

```
10 ?"JONES,P.J          431-25-6277          1420.00          258.74"
```

Zur Verkürzung der Anweisung könnten wir statt der Leerstellen die Funktion SPC verwenden. Wir müßten den Abstand der Spalten zueinander bestimmen und jeweils in die SPC-Funktion einsetzen:

```
10 ?"JONES, P.J";SPC(17);"431-25-6277";SPC(5);"1420.00";SPC(9);"258.74"
```

Am einfachsten aber geht es mit der Funktion TAB, da wir die Spaltenpositionen 0, 16, 32, 48 kennen:

```
10 ?"JONES,P.J";TAB(16);"431-25-6277";TAB(32);"1420.00";TAB(48);"258.74"
```

Beachten Sie folgendes: In den letzten PRINT-Anweisungen haben wir auch die darzustellenden Zahlen als Zeichenketten, das heißt zwischen Anführungszeichen,

ausgedrückt. Hierdurch erfolgt die Zahlendarstellung genauso, wie sie in der PRINT-Anweisung steht. Stellt man Zahlen nicht als Zeichenketten dar, dann werden störende Regeln wirksam: Vor jeder Zahl wird eine Leerstelle zur Aufnahme eines Minuszeichens gelassen, und nach einem Dezimalpunkt werden keine Nullen dargestellt (s. Kap. 5).

KURSORPOSITION: POS

Die Funktion POS meldet die augenblickliche Stellung des Kursors in einer Bildschirmzeile. Die gemeldete Zahl gibt die Spalte an, in der der Kursor steht. POS ( ) wird mit dem Scheinargument 0 in der Parameterliste einer PRINT-Anweisung verwendet:

```
10 PRINT "POSITION DES KURSORS:";POS(0)
```

Diese Anweisung kann auch in direktem Dialog erprobt werden:

```
10 PRINT "POSITION DES KURSORS:";POS(0)
POSITION DES KURSORS: 21
```

In diesem Fall befand sich der Kursor auf Spalte 21. Der PRINT-Befehl gibt zunächst den Text "Position des Kursors:" aus, danach die Meldung von POS. Fügen Sie noch einige Leerstellen an den Doppelpunkt an, um POS zu einer anderen Meldung zu veranlassen.

DATENEINGABE: INPUT

Eine INPUT-Anweisung läßt den Computer auf Eingaben vom Tastenfeld warten; ehe diese Eingabe nicht erfolgt ist, ereignet sich nichts weiteres im Computer

Der INPUT-Anweisung folgt eine Liste von Variablennamen. Eingaben am Tastenfeld werden der Reihe nach diesen Variablen zugewiesen. Der Typ der Variablen und der Typ der Tastenfeldeingabe müssen übereinstimmen: Für Zeichenketten-Variable mit einem $ am Ende darf nur Text (aus bis zu 80 Zeichen) eingegeben werden; für Gleitkomma- und Ganzzahl-Variable nur Zahlen.

```
10 INPUT A$
20 ?A$
30 GOTO 10
```

Zur Ausführung der INPUT-Anweisung stellt der Computer ein Fragezeichen auf dem Bildschirm dar und wartet auf Ihre Eingabe. Das oben stehende Programm stellt jeden Text dar, den Sie nach dem Fragezeichen eingeben: Zunächst liest INPUT Ihren Text in die Variable A$, dann stellt eine PRINT-Anweisung in Zeile 20 diesen Text dar.

Im folgenden Beispiel erwartet eine INPUT-Anweisung der Reihe nach die Eingabe einer Zeichenkette A$, einer Ganzzahl A und einer Gleitkommazahl A%:

```
10 INPUT A$,A,A%
20 ?A$,A,A%
30 GOTO 10
```

Nach dem Fragezeichen ? auf dem Bildschirm wird von Ihnen eine Eingabe erwartet, die in Reihenfolge und Form mit der Variablenangabe in INPUT übereinstimmt: Zuerst eine Zeichenkette, dann ein Komma, dann eine Ganzzahl, ein weiteres Komma und eine Gleitkommazahl. Jede Abweichung von dieser Reihenfolge löst eine Fehlermeldung in Form eines doppelten Fragezeichens ?? aus. Nach dieser Fehlermeldung haben Sie Gelegenheit, die Eingabe in der richtigen Form zu wiederholen.

```
RUN10

? ABC                          :UNVOLLSTAENDIGE EINGABE
?? ABC,100,100.5               :EINGABE MIT FORMATFEHLER
?REDO FROM START
? ABC,100.5,100                :EINGABE OHNE FEHLER
ABC          100.5       100
```

Wie wir früher besprochen haben, kann jede Ganzzahl durch eine Gleitkommavariable dargestellt werden. Sie dürfen aber nicht eine Gleitkommazahl für eine Ganzzahlvariable eingeben. Außerdem gilt folgendes: Sie dürfen eine Zahl für eine Zeichenketten-Variable eingeben, nicht aber eine Zeichenkette für eine Zahlen-Variable.

Wie Sie sehen, sind verschiedene Erlebnisse mit einer INPUT-Anweisung möglich. Diese Erlebnisse können Sie keinem Programmbenutzer zumuten, der sich um seine Arbeit kümmern muß und nicht um die Deutung von Fehlermeldungen. In Kapitel 5 beschreiben wir daher Eingabeprogramme, bei deren Konzipierung jede mögliche Fehleingabe eines Benutzers bedacht und Benutzer-verständliche Rückfragen gestellt werden.

Aber einen Trick mit der INPUT-Anweisung sollten Sie sich merken: Statt des Fragezeichens ? kann INPUT auch einen Text zur Darstellung bringen, beispielsweise mit der Aufforderung zur Eingabe einer Zahl oder eines Textes. Die Bildschirmnachricht muß wie eine Zeichenkette zwischen Anführungszeichen angegeben werden und mit einem Semikolon abgeschlossen werden. Hier ein Beispiel:

```
10 PRINT "POSITION DES KURSORS:";POS(0)
POSITION DES KURSORS: 21
```

```
10 INPUT "BITTE DIE ZAHL 1 EINGEBEN";N
20 IF N<>1 THEN GOTO 50
30 PRINT "RICHTIG"
40 GOTO 40
50 PRINT "FALSCH:"
60 GOTO 10
70 GOTO 70
```

Dieses Programm stellt auf dem Bildschirm die Aufforderung "Bitte die Zahl 1 eingeben" dar und wartet auf Ihre Eingabe. Bei falscher Eingabe erfolgt die Meldung "FALSCH:" mit der Möglichkeit für eine richtige Eingabe, die dann mit der Nachricht "RICHTIG" belohnt wird.

Eine Textausgabe mit INPUT erlaubt, dem Benutzer Bedeutung und Form einer erwarteten Eingabe mitzuteilen.

ZEICHENEINGABE: GET

Die GET-Anweisung ermöglicht einem Programm, das Tastenfeld auf die Eingabe eines einzelnen Zeichens durch den Programmbenutzer abzufragen, und ein eingegebenes Zeichen in einer Variablen des Programms zu speichern. Der Programmbenutzer muß seine Zeicheneingabe nicht mit einem RETURN abschließen. Jedes vom CBM-Computer benutzte Zeichen kann eingelesen werden. Für die Eingabe nichtnumerischer Zeichen steht nach einer GET-Anweisung eine Zeichenketten-Variable:

```
10 GET A$
20 ?A$
30 GOTO 10
```

Wenn Sie dieses Programm mit RUN zum Laufen bringen, verschwinden alle Anzeigen nach oben aus dem Bildschirm. GET wartet nicht auf eine Zeicheneingabe. Ein gerade im Tastenfeldspeicher vorhandenes Zeichen wird in der Programmschleife endlos oft dargestellt. Machen Sie einen Versuch mit der Eingabe eines Buchstabens.

Im folgenden Beispiel wartet die GET-Anweisung auf die Eingabe des Buchstabens X. Kein anderer Buchstabe wird von diesem Programm gelesen.

```
10 GET A$
20 IF A$<>"X" THEN GOTO 10
30 ?A$
40 GOTO 10
```

GET kann auch programmiert werden, auf beliebige Eingaben zu warten. Die Programmlogik benutzt hierfür folgende Eigenschaft einer Anweisung der Form GET A$: Keine Eingabe, d.h. ein leerer Tastenfeldspeicher wird als leere Zeichenkette interpretiert. Dieser Zustand wird im Programm durch A$ = " " ausgedrückt:

```
10 GET A$
20 IF A$="" THEN GOTO 10
30 ?A$
40 GOTO 10
```

Bei Angabe einer Ganzzahl- oder Gleitkomma-Variablen erwartet eine GET-Anweisung die Eingabe einer Zahl. Solange keine Eingabe erfolgt, wird diesen Variablen der Zahlenwert 0 zugewiesen. Eine Programmlogik hat also keine Möglichkeit, aus einer mit GET gelesenen Null zu erkennen, ob sie eine Eingabe darstellt oder nicht. GET-Anweisungen werden deshalb gewöhnlich nur für nicht-numerische Zeichen verwendet.

In Programmen wird von der GET-Anweisung dann am häufigsten Gebrauch gemacht, wenn ein Dialog zwischen Programm und Benutzer erforderlich ist. Die folgenden Programmzeilen warten auf die Eingabe eines J für JA durch den Benutzer:

```
10 PRINT "BIST DU DA? GIB MIR J EIN FUER JA"
20 GET A$
30 IF A$<>"J" THEN GOTO 20
40 PRINT "GUT! LASS UNS WEITERMACHEN."
```

Die Benutzereingabe wird nicht dargestellt. Ändern Sie das Programm derart, daß der Benutzer eine Darstellung als Quittung für seine Eingabe erhält.

ZUGRIFF AUF SPEICHERPLÄTZE: PEEK, POKE

CBM-Computer können bis zu 65,536 einzeln adressierbare Speicherplätze haben, von denen jeder eine Zahl zwischen 0 und 255 entsprechend den Möglichkeiten von 8 Bit ($2^8 - 1 = 255$) speichern kann. Alle Anweisungen und Daten werden in derartige Zahlen umgewandelt und gespeichert.

Eine PEEK-Anweisung ermöglicht, die in jeder Speicherstelle eines CBM-Computers gespeicherte Zahl zu lesen:

```
10 A%=PEEK(200)
```

Diese Anweisung weist der Variablen A% den Inhalt des Speicherplatzes 200 zu. Das Argument in PEEK kann eine Zahl sein, wie oben, eine Ganzzahl-Variable oder ein Ganzzahl-Ausdruck; in jedem Fall muß sich ein Zahlenwert als Adresse eines Speicherplatzes ergeben.

Die POKE-Anweisung ermöglicht, Daten in einen Speicherplatz des CBM-Computers zu schreiben:

```
20 POKE 8000,A%
```

Diese Anweisung speichert den Inhalt der Variablen A% im Speicherplatz mit der Adresse 8000. Anstelle einer Variablen wie A%, kann eine Zahl oder ein Ausdruck stehen, deren Zahlenwert zwischen 0 und 255 liegt. Gleitkommazahlen werden in Ganzzahlen umgewandelt.

Eine PEEK-Anweisung läßt sich auf Lese/Schreib-Speicher (RAM) und reine Lese-Speicher (ROM) anwenden. Eine POKE-Anweisung ist dagegen nur bei Lese/Schreib-Speichern anwendbar, was einfach zu verstehen ist.

PROGRAMMENDE: END, STOP

Ein Programmablauf wird mit den Anweisungen END und STOP zum Halt gebracht. Zur Fortsetzung des Programmablaufs geben Sie CONT über das Tastenfeld ein. Die Anweisungen END oder STOP müssen nicht unbedingt in Ihrem Programm stehen, stellen jedoch einen eindeutigen Programmabschluß dar.

In einigen Programmbeispielen dieses Kapitels haben wir GOTO-Anweisungen benutzt, die auf sich selbst verzweigen und so das Programm "auf der Stelle treten lassen":

```
50 GOTO 50
```

Eine derartige Anweisung wird endlos lange ausgeführt. Stattdessen könnten wir auch eine STOP-Anweisung verwenden, nach deren Ausführung folgende Meldung auf dem Bildschirm erscheint:

```
BREAK IN XXXX
READY
```

Es erscheint die Zeilennummer XXXX derjenigen STOP-Anweisung, die den Programmabbruch verursachte. Sind in Ihrem Programm mehrere STOP-Anweisungen vorgesehen, dann zeigt XXXX an, an welcher Stelle des Programms der Abbruch erfolgte.

FUNKTIONEN

Ein anderes Element der CBM BASIC-Sprache sind Funktionen, die etwas wie Variable aussehen und mehr wie BASIC-Anweisungen wirken.

Hier ein einfaches Beispiel für eine Funktion:

```
10 A=SQR(B)
```

Der Variablen A soll die Quadratwurzel der Variablen B zugewiesen werden. SQR führt die erforderliche Wurzelbildung aus. Und hier eine Funktion für Zeichenketten:

```
20 C$=LEFT$(D$,2)
```

Der Variablen C$ sollen die ersten beiden Zeichen in der Zeichenkette D$ zugewiesen werden. Die Funktion LEFT$ führt die Zeichenabtrennung aus.

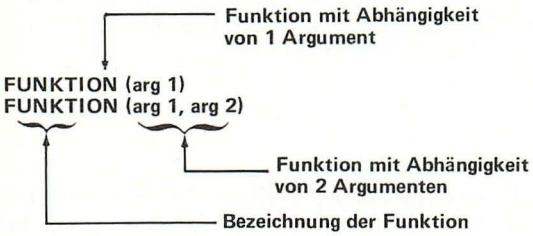
Variable und Konstante können überall in einer BASIC-Anweisung durch Funktionen ersetzt werden, aber eine Funktion muß immer auf der rechten Seite

eines Gleichheitszeichens stehen: A = SQR (B) ist möglich, nicht aber SQR (B) = A.

Wir haben früher bereits Funktionen benutzt: SPC, TAB, POS (die alle in PRINT-Anweisungen auftreten) und PEEK.

Die folgende Darstellung zeigt Ihnen den Gebrauch von Funktionen. Eine vollständige Beschreibung aller Funktionen in CBM BASIC finden Sie in Kapitel 8.

Eine Funktion besteht aus einem Funktionsnamen, angedeutet durch Buchstaben wie COS für KOSINUS, gefolgt von zwei Klammern, in denen ein oder zwei Argumente stehen:



Nur wenige Funktionen sind abhängig von drei Argumenten.

Jedes Argument einer Funktion kann eine Konstante, eine Variable oder ein Ausdruck sein. Eine Funktion in einer BASIC-Anweisung wird vor jedem anderen Operator ausgewertet. Jede Funktion in einer BASIC-Anweisung liefert schließlich als Ergebnis eine Zahl oder Zeichenkette. Erst dann werden die übrigen Operationen in einer BASIC-Anweisung ausgeführt:

```
10 B=24.7*(SQR(C)+5)-SIN(0.2+D)
```

Die Funktionen SQR und SIN in obenstehender Programmzeile werden zuerst ausgewertet: Nehmen wir an, daß SQR (C) = 6.72 und SIN (0.2 + D) = 0.625. Die Anweisung reduziert sich dann wir folgt:

```
10 B=24.7*(6.72+5)-0.625
```

In dieser Form wird die Anweisung schließlich ausgeführt.

ARITHMETISCHE FUNKTIONEN

Nachfolgend eine Liste arithmetischer Funktionen in CBM BASIC:

```
INT( )  VOM ARGUMENT WIRD DER GANZZAHLIGE ANTEIL ZURUECKGEMELDET
SGN( )  VOM ARGUMENT WIRD DAS VORZEICHEN WIE FOLGT GEMELDET ALS:
        -1 FUER NEGATIVE,+1 FUER POSITIVE,0 FUER ZAHLEN MIT WERT 0
ABS( )  VOM ARGUMENT WIRD DER ABSOLUTWERT (BETRAG) ZURUECKGEMELDET
SQR( )  VOM ARGUMENT WIRD DIE QUADRAT-WURZEL GEBILDET
EXP( )  DAS ARGUMENT DIENT ALS EXPONENT ZUR ZAHL E=2.71828...
LOG( )  VOM ARGUMENT WIRD DER NATUERLICHE LOGARITHMUS GEBILDET
RND( )  DAS ARGUMENT DIENT ZUR AUSWAHL EINER FOLGE VON ZUFALLSZAHLEN
SIN( ) ⎫                                                  SINUS
COS( ) ⎪                                                  KOSINUS
TAN( ) ⎬VOM ARGUMENT IN RADIAN WIRD DER TRIGONOMETRISCHE  TANGENS
ATN( ) ⎭                                                  ARKUSTANGENS
                                                          GEBILDET
```

Hier ein Beispiel für ihre Anwendung:

```
10 A=2.743
20 B=INT(A)+7
30 ?B
40 STOP
```

Ein Programmlauf liefert das Ergebnis 9. Der ganzzahlige Anteil von A = INT (A) = 2. Wenn Sie Zeile 10 in eine INPUT-Anweisung und Zeile 40 in eine GOTO 10-Anweisung umwandeln, können Sie die Arbeitsweise der Ganzzahl-Funktion INT untersuchen.

Hier ein verwickelteres Beispiel für arithmetische Funktionen:

```
10 INPUT A,B
20 IF LOG(A)<0 THEN A=1/A
30 ?SQR(A)*EXP(B)
40 GOTO 10
```

Das Argument einer Funktion kann ein Ausdruck sein, der selbst Funktionen enthält:

```
30 ?SQR(A*EXP(B)+3)
```

In diesem Beispiel wird zur Wurzelbildung zunächst das Argument A* EXP (8) + 3 ausgewertet, und von der resultierenden Zahl die Wurzel, SQR, gebildet.

Wenn Sie keine Anwendung für arithmetische Funktionen haben, können Sie diesen Abschnitt überspringen. Auf jeden Fall ist aber der folgende Abschnitt über Funktionen für Zeichenketten von allgemeiner Wichtigkeit.

ZEICHENKETTEN-FUNKTIONEN

Funktionen für Zeichenketten ermöglichen, auf vielfältige Weise Zeichenketten zu bearbeiten und zu verändern. Hier eine Liste der Zeichenketten-Funktionen in CBM BASIC:

```
STR$( )  STELLT EINE (ZAHL) ALS ZEICHENKETTE DAR
VAL( )   STELLT FUEHRENDE ZIFFERN IN EINER (ZEICHENKETTE) DAR
CHR$( )  STELLT DAS ZUM (ASCII-KODE) GEHOERENDE SYMBOL DAR
ASC( )   STELLT DEN ZUM (SYMBOL) GEHOERENDEN ASCII-KODE DAR
LEN( )   ZAEHLT DIE ZAHL DER SYMBOLE IN EINER (ZEICHENKETTE)
LEFT$    ,              LINKEN
MID$     |BLENDET DIE MITTLEREN SYMBOLE IN EINER ZEICHENKETTE AUS
RIGHT$   /              RECHTEN

GENAUERE ANGABEN: KAPITEL 8  UND  ANHANG A,TABELLE A-4
```

Mit diesen Funktionen können Sie die Länge einer Zeichenkette bestimmen, Teile aus einer Zeichenkette herausschneiden, und einen Datentyp ändern, z.B. eine Zahl als Zeichenkette darstellen.

In Kapitel 8 finden Sie für jede dieser Funktionen eine genaue Beschreibung ihrer Wirkung und ihrer Anwendung.

SYSTEM-FUNKTIONEN

Für die Vollständigkeit der Aufzählung von Funktionen in CBM BASIC wird hier eine Liste von Funktionen angegeben, mit denen man an Daten und Programme im CBM-Computer gelangen kann, die im übrigen der Programmiersprache BASIC nicht zugänglich sind.

Derartige Funktionen werden für Sie erst dann von Bedeutung sein, wenn Ihnen die in BASIC gegebenen Möglichkeiten nicht mehr ausreichen.

Folgende Funktion jedoch werden Sie bald verwenden: Die im CBM-Com-

puter gebildete Tageszeit TI$, mit der Sie Ihre Programmausdrucke kennzeichnen können, wenn sich während einer Programmentwicklung im Tagesverlauf zahlreiche Versionen des gleichen Programms ergeben.

Eine genaue Beschreibung der folgenden System-Funktionen finden Sie in Kapitel 8.

```
TI$,TI   LIEST DIE COMPUTER-INTERNE UHR
PEEK     LIEST DEN INHALT EINES SPEICHERPLATZES
FRE      MELDET NOCH VERFUEGBAREN SPEICHERPLATZ
SYS      SPRINGT IN EIN MASCHINENPROGRAMM
USR      UEBERGIBT PARAMETER AN EIN MASCHINENPROGRAMM
```

BENUTZER-FUNKTIONEN: DEF FN

Neben den Standard-Funktionen in CBM BASIC können Sie auch eigene arithmetische Funktionen definieren, sogenannte Benutzer-Funktionen. Die Funktionen für Zeichenketten können nicht um Benutzer-Funktionen erweitert werden. Im CBM-Computer wird mit DEF FN die Eingabe einer Benutzer-Funktion angezeigt:

```
10 DEFFNP(X)=100*X
20 INPUT A
30 ?A,FNP(A)
40 GOTO 20
```

An die Mitteilung DEF FN muß sich unmittelbar eine Gleitkomma-Variable anschließen, die zusammen mit FN den Namen der Benutzer-Funktion bildet. Im oben genannten Beispiel also FNT. Lautete die Variable AB, dann wäre der Name der Benutzer-Funktion FNAB.

Das Argument der Benutzer-Funktion steht in Klammern nach dem Namen der Funktion, während rechts des Gleichheitszeichens der vom Benutzer definierte funktionelle Zusammenhang steht.

Die in einer DEF FN-Anweisung verwendete Variable (im Beispiel X) kann im übrigen Programm frei verwendet werden, d.h. ist nicht durch die Benutzer-Funktion verbraucht.

KAPITEL 5

Wahrnehmen aller Fähigkeiten eines CBM Computers

Dieses Kapitel beschreibt gerätetechnische und programmtechnische Fähigkeiten von CBM-Computern.

GERÄTETECHNISCHE FÄHIGKEITEN

VERARBEITUNG SCHNELLER TASTENANSCHLÄGE

Wenn Sie zwei oder mehr Tasten gleichzeitig bedienen, oder wenn Sie eine zweite Taste drücken, noch ehe die vorangegangene Tasteneingabe zur Anzeige gekommen ist – dann gehen im allgemeinen Tastenanschläge verloren, wenn Ihr Computer kein Tastengedächtnis besitzt. Glücklicherweise sind alle CBM-Computer mit einem Tastengedächtnis ausgerüstet.

Das Tastengedächtnis merkt sich Tastenanschläge, die während der Bearbeitung der letzten Tasteneingabe eintreffen. Das Gedächtnis speichert Tasteneingaben bis zu ihrer Bearbeitung durch den Computer. Ohne dieses Gedächtnis würden schnelle Tasteneingaben verloren gehen. Wenn zum Beispiel Tastenanschlag #2 erfolgt, noch bevor Tastenanschlag #1 bearbeitet ist, dann speichert der CBM-Computer Tastenanschlag #2, bis Anschlag #1 endgültig bearbeitet ist. Dann wird Tastenanschlag #2 aus dem Speicher geholt und ebenfalls bearbeitet.

Das Tastengedächtnis ist eine sehr nützliche Einrichtung in CBM-Computern, da es sehr schnelle Dateneingabe ohne gelegentlichen Verlust von Tastenanschlägen erlaubt.

TASTENFELDSPEICHER

Alle CBM-Computer verfügen über einen Speicher für 10 vom Tastenfeld eingegebene Zeichen.

Lassen Sie zur Illustration die endgültige Version des Programms BLANKET aus Figur 5-1 ablaufen. Drücken Sie bis zu zehn Tasten, während die erste Bildschirm-

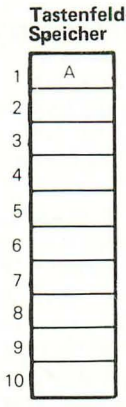
ausgabe erzeugt wird, und lehnen sich zurück und warten. Jedes der zehn eingegebenen Zeichen wird vom Speicher geholt und der Reihe nach vom BLANKET-Programm zur Anzeige gebracht.

Figur 5 - 1 Programm BLANKET

```
10 REM ******* B L A N K E T *******
20 REM ZEILENWEISE DARSTELLUNG EINES
30 REM EINGEGEBENEN SYMBOLS
40 REM ****************************
90 PRINT"BITTE SYMBOL EINGEBEN ODER MIT RETURN BEENDEN"
100 GET C$: IF C$="" GOTO 100
105 IF  C$=CHR$(13)  GOTO 170
110 PRINT"■";              :REM LOESCHEN DES BILDSCHIRMS
120 FOR I=1 TO 960         :REM 960/80=12 ZEILEN
130 PRINT C$;
140 NEXT
150 PRINT "PFUI!"
160 GOTO 90
170 END
```

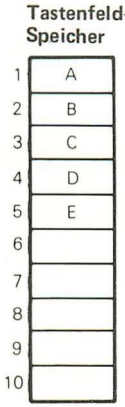
Lassen Sie uns diesen Vorgang genauer ansehen.

Eine erste Tasteneingabe geht in die erste Stelle des 10-Zeichen-Speichers. War die Eingabe ein A, dann sieht es so aus:



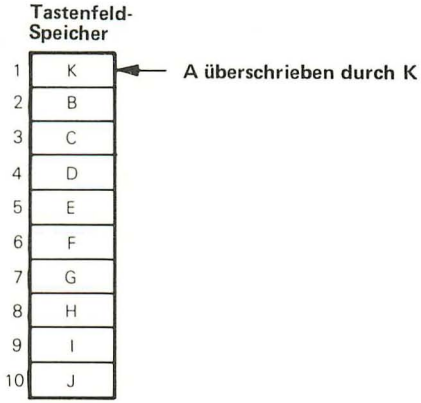
Der CBM-Computer verfolgt die Zahl der Tasteneingaben in den Speicher und kennt das nächste darzustellende Zeichen. Jeder GET-Befehl, mit dem ein Zeichen aus dem Speicher geholt wird, erhöht zugleich einen Zeiger auf die nächste Speicherstelle.

Wenn Sie weitere Tasten drücken, während das A noch angezeigt wird, dann werden die eingegebenen Zeichen in den nächsten freien Speicherstellen gespeichert. Nehmen Sie an, Sie geben ein A ein, und noch während das A zur Anzeige gebracht wird, geben Sie B, C, D und E ein. Alle diese Zeichen werden im Tastenfeldspeicher gespeichert:



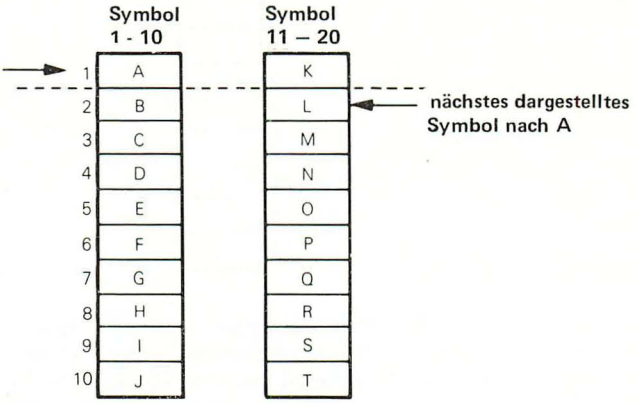
Lassen Sie das BLANKET Programm weiterhin laufen, dann wird es nacheinander alle im Tastenfeldspeicher gespeicherten Zeichen zur Anzeige bringen. Nach Darstellung des A wird es das B holen und über 20 Zeilen darstellen, dann das C, usw.

Wenn Sie jedoch mehr als zehn Zeichen eingeben, dann fängt die Speicherung bei allen CBM-Modellen mit Ausnahme der CBM 8000-Serie wieder von vorne an. Wenn Sie beispielsweise die ersten 11 Buchstaben des Alphabets (A - K) eingeben, dann werden die ersten zehn Buchstaben in die zehn Speicherstellen geschrieben, der Buchstabe K jedoch wieder in die erste Speicherstelle, wobei er den dortigen Buchstaben A überschreibt:



Wenn das Programm jetzt die Anzeige des A abgeschlossen hat, kehrt es zurück zum Tastenfeldspeicher, um das nächste Zeichen zu holen. Die letzte Zeicheneingabe war in Speicherstelle 1 erfolgt, von dort hatte der Computer aber gerade ein Zeichen abgeholt, so daß er annimmt, der Tastenfeldspeicher sei leer. Die Eingabe von genau elf Zeichen, oder Vielfachen davon, löst keine weiteren automatischen Anzeigen durch das Programm BLANKET aus.

Die Eingabe von 12 bis 20 Zeichen führt zur Anzeige des ersten Zeichens und dann der Zeichen 12, 13, usw. Zum Beispiel: geben Sie ein A ein, und noch während das A dargestellt wird, die Zeichen B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S und T.



Die Reihenfolge der Anzeige wird sein: A, L, M, N, O, P, Q, R, S, T.

Computer der Serie CBM 8000 weisen Zeichen zurück, die das Speichervermögen des Tastenfeldspeichers überschreiten.

LEEREN DES TASTENFELDSPEICHERS VOR EINEM GET-BEFEHL

Der Tastenfeldspeicher ist eine leichte Überraschung, gewöhnlich eine angenehme. Beim Programm BLANKET können Sie bis zu zehn Zeichen auf Vorrat speichern, ohne jedesmal auf die Aufforderung des Programms ZEICHEN EINGEBEN warten zu müssen. Aber der Tastenfeldspeicher kann auch eine unerwartete Überraschung bringen. Gelegentlich kann eine Tasteneingabe das Programm veranlassen, ein ungewünschtes Zeichen aus dem Tastenfeldspeicher zu holen. Um so etwas zu vermeiden, können Sie wie folgt über eine Programmschleife den Tastenfeldspeicher leeren, ehe die gewünschte Tasteneingabe aus dem Speicher abgeholt wird:

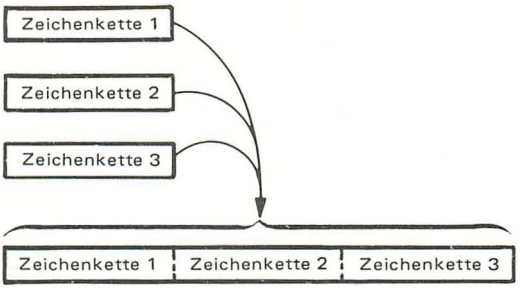
```
95  FOR I=1 TO 10:GET C$:NEXT I:REM TASTENFELDSPEICHER LEEREN
100 GET C$:IF C$="" GOTO 100
```

Die Anweisung in Zeile 95 leert den Tastenfeldspeicher, indem sie alle zehn möglichen Zeichen aus dem Speicher liest.

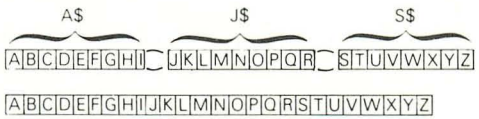
Erweitern Sie das Programm BLANKET um Zeile 95 wie oben angegeben. Drücken Sie jetzt irgend eine Tastenkombination, während noch ein Zeichen angezeigt wird. Die GET-Schleife wird jedes gespeicherte Zeichen auslesen und unterdrücken, so daß Sie keine fortlaufende automatische Anzeige erhalten.

VERBINDEN VON ZEICHENKETTEN

Der CBM-Computer erlaubt alphabetische, graphische und numerische Zeichen innerhalb von Zeichenketten. Beim Arbeiten mit Zeichenketten kann es erforderlich sein, eine einzelne Zeichenkette aus kürzeren Zeichenketten zu bilden, die wie bei einer Kette Ende an Ende zusammengefügt wurden:



Nehmen Sie an, wir wollen eine lange Zeichenkette, Z $, bilden, die das Alphabet A bis Z enthält. Hierzu können wir das letzte Zeichen der Kette A $, wie unten dargestellt, mit dem ersten Zeichen der Kette I $, und das letzte Zeichen von J$ mit dem ersten von S $, verbinden:



Der arithmetische Operator "+" addiert den Zahlenwert numerischer Variablen, bewirkt jedoch eine Verkettung, wenn er auf Zeichenketten angewandt wird. Tabelle 5-1 gibt eine Übersicht über die Wirkung des "+"-Operators bei Zahlen und Zeichenketten.

Tabelle 5 - 1. Formen der Addition

Vorzeichen	Typ	Beispiel	Formen der Addition	Ergebnis
+	Zahlen	P = 2 + 3	2 + 3	P = 5
+	numerische Variable	Q = T + S T = 1 2 3 4 5 S = 1 1 1 1 1	12345 +11111 23456	Q = 23456
+	alphanumerische Zeichenketten	R$ = A$ + F$ A$ = A B C D E F$ = F G H I J	A B C D E ‿ F G H I J	R$ = A B C D E F G H I J
+	numerische Zeichenketten	Q$ = T$ + S$ T$ = 1 2 3 4 5 S$ = 1 1 1 1 1	1 2 3 4 5 ‿ 1 1 1 1 1	Q$ = 1 2 3 4 5 1 1 1 1 1

Beachten Sie jedoch: Zeichenketten können nicht in gleicher Weise getrennt oder in Stücke zerlegt werden, wie sie verkettet wurden; es gibt keine "Subtraktion" zur "Addition". Es wäre zum Beispiel falsch, die Zeichenkette X$ als Inhalt der Ketten I$ und S$ von Z$ subtrahiert:

```
X$=Z$-J$-S$ ◄— unzulässige Verkettung
```

Versuchen Sie es. Geben Sie die Symbole A$, J$, S$ und X$=Z$–A$ in Ihren CBM-Computer wie unten gezeigt. Die Antwort wird ein
?TYPE MISMATCH ERROR IN LINE 50 (unerlaubtes Zeichen in Zeile 50) sein:

```
10 A$="ABCDEFGHI"
20 J$="JKLMNOPQR"
30 S$="STUVWXYZ"
40 Z$=A$+J$+S$
50 X$=Z$-J$-S$
60 PRINT

RUN


?TYPE MISMATCH ERROR IN 50
```

Der einzig zulässige arithmetische Operator für Zeichenketten ist das Additionszeichen (+). Die übrigen arithmetischen Zeichen (–, x, /) bewirken nichts, jedoch können die Boole'schen Operatoren (<, >, =) zum Vergleich von Zeichenketten verwendet werden.

Das richtige Verfahren, aus einer längeren Zeichenkette Teile herauszutrennen, besteht in der Verwendung von Zeichenketten-Funktionen. Mit den Funktionen LEFT$, MID$ und RIGHT$ ist es möglich, jeden gewünschten Teil einer Zeichenkette abzutrennen. Die Buchstaben J bis Z im Beispiel unserer Zeichenkette Z$ können wie folgt abgetrennt werden:

```
50 X$=RIGHT$(Z$,17)
   X$ = RIGHT$(ABCDEFGHIJKLMNOPQRSTUVWXYZ,17)
   X$ = JKLMNOPQRSTUVWXYZ
```

Oder aber man bildet die gewünschte Zeichenkette durch Verkettung von J$ und S$:

```
50 X$=J$+S$
   X$ = JKLMNOPQR+STUVWXYZ
   X$ = JKLMNOPQRSTUVWXYZ
```

VERKETTUNG BEI DRUCKER/BILDSCHIRMAUSGABEN

Wenn Sie Zeichenketten lediglich auf Drucker oder Bildschirm ausgeben wollen, dann verwenden Sie zu deren Verkettung den PRINT-Befehl mit Semikolon (;) als Trennsymbol zwischen den Zeichenketten:

```
PRINT A$;J$;S$

ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Das Ergebnis der Verkettung (Zeichenkette A bis Z) wird nirgendwo im CBM-Computer abgespeichert.

GRAPHISCHE ZEICHENKETTEN

Graphische Zeichenketten werden wie alphanumerische Ketten verbunden. Dies ist ein nützliches Verfahren zur Herstellung von Bildern und Diagrammen.

NUMERISCHE ZEICHENKETTEN

Eine numerische Zeichenkette kann als Zahl gelesen werden. Numerische Zeichenketten können auf zwei Weisen gebildet werden, mit leicht verschiedenen Ergebnissen.

Wird eine numerische Zeichenkette (im Beispiel unten: T$) mit Hilfe der Funktion STR$ durch Zuweisung zu einer Variablen (im Beispiel unten: AB) gebildet, dann wird das Vorzeichen der Zahl (im Beispiel unten: 12345) mit in die numerische Zeichenkette übertragen. Eine Leerstelle steht für ein "+"-Zeichen, ein "−" für ein Minuszeichen. Das wird im folgenden Programm gezeigt:

```
10 AB=12335
20 T$=STR$(AB)
30 PRINT"AB=";AB
40 PRINT"T$=";T$

RUN

AB= 12345
T$= 12345
```

Wird jedoch eine Zahl in "Anführungszeichen" eingegeben (oder mit Hilfe der Befehle INPUT, GET oder READ eine Zeichenkette gelesen), dann wird die numerische Zeichenkette wie jede andere alphabetische oder graphische Zeichenkette behandelt. Es wird also keine Leerstelle für ein Pluszeichen eingefügt. Dies wird am folgenden Programm gezeigt:

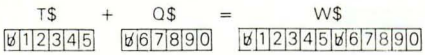
```
10 AB=12345
20 T$="12345"
30 PRINT"AB=";AB
40 PRINT"T$=";T$

RUN

AB= 12345
T$=12345
```

Lassen Sie uns jetzt zwei numerische Zeichenketten, T$ und Q$, mit einander verbinden, um die Kette W$ zu bilden. W$ soll die zehn Ziffern 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 enthalten. Hier eine Möglichkeit:

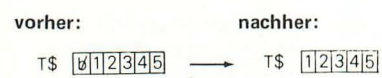
```
10 T=12345
20 Q=67890
30 T$=STR$(T)
40 Q$=STR$(Q)
50 W$=T$+Q$
60 PRINT"W$=";W$

RUN

W$= 12345 67890
```

Warum die Leerstellen vor der 1 und vor der 6? T$ und Q$ waren ursprünglich positive numerische Variable, T und Q; als T und Q von Zahlen in Zeichenketten umgewandelt wurden, blieb das Pluszeichen (Leerstelle) erhalten.

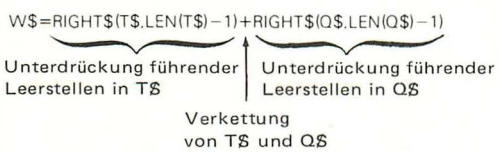
Daher enthält die aus T$ und Q$ entstandene neue Kette W$ zwei Leerstellen: am Anfang und in der Mitte:



Um diese Leerstellen los zu werden, schauen wir noch einmal auf die Zeichenketten T$ und Q$. Die einzigen Symbole, die wir in der neuen Kette W$ wünschen, sind die Ziffern rechts der Vorzeichen. Mit den Befehlen LEFT$, MID$ und RIGHT$ können wir jedes Zeichen oder jede Zeichengruppe aus einer Zeichenkette auswählen. Was wir wollen, sind alle Zahlen rechts des Vorzeichens : das eine Leerstelle ist für ein Pluszeichen, oder ein "–" für ein Minuszeichen.
T$=RIGHT(T$',LEN(T$)-1) tut uns den Gefallen:



Da das erste gewünschte Zeichen an der zweiten Stelle der Zeichenkette steht, weisen wir unseren CBM-Computer an, nur die mit Stelle 2 beginnenden Zeichen zu verwenden (LEN=Länge der Kette vom Ende gezählt=6,LEN(T$)–1=Stelle 5 vom Ende gezählt=Stelle 2 vom Anfang gezählt). Wir können T$ und Q$ verketten und die führenden Leerstellen unterdrücken mit einer einzigen Anweisung:



Unser verbessertes Programmbeispiel nimmt dann folgende Form an:

```
10 T=12345
   T = 12345

20 Q=67890
   Q = 67890

30 T$=STR$(T)
   T$ = 12345

40 Q$=STR$(Q)
   Q$ = 67890

50 W$=RIGHT$(T$,LEN(T$)-1)+RIGHT$(LEN(Q$)-1)
   W$ = RIGHT$(T$,6-1)          +RIGHT$(Q$,6-1)
   W$ = RIGHT$(T$,5)            +RIGHT$(Q$,5)
   W$ = T$ 12345                +Q$ 67890
   W$ = 1234567890

60 PRINT "W$=";W$

RUN

W$=1234567890
```

Beachten Sie in obigem Beispiel, daß Zeile 50 nicht nach negativen Zahlen fragt. Sind beide Zahlen negativ, dann sollte das führende Zeichen von T$ nicht unterdrückt werden; hierdurch kann das Minuszeichen vor der resultierenden Zahl W$ erscheinen. Haben zwei numerische Zeichenketten unterschiedliche Vorzeichen, dann sollten sie nicht verkettet werden.

EIN/AUSGABE-PROGRAMMIERUNG

Ein Anfänger in der Programmierung entdeckt schnell, daß die Ein/Ausgabe-Abschnitte eines Programms voller Tricks sind.

Nahezu jedes Programm benötigt Daten, die über das Tastenfeld eingegeben werden müssen. Werden einige wenige INPUT (EINGABE)-Befehle ausreichen? In den meisten Fällen ist die Antwort NEIN. Und wenn dann der Bedienende falsche Tasten gedrückt hat oder entdeckt, daß falsche Daten eingegeben wurden? Ein brauchbares Programm muß berücksichtigen, daß ein Mensch den Computer bedient und sehr wahrscheinlich menschliche Fehler gemacht werden.

Ergebnisse können ebensowenig einfach dargestellt oder ausgedruckt werden, indem eine Anzahl von PRINT (DRUCK)-Befehlen verwendet wird. Die Darstellungen sollen von einem Menschen gelesen werden. Wenn Ergebnisdarstellungen nicht sorgfältig geplant sind, werden sie schwierig zu lesen sein; als Folge könnte eine Information falsch aufgefaßt oder völlig übersehen werden.

Glücklicherweise verfügt CBM BASIC über genügend Möglichkeiten, Eingabe und Ausgabe einfach und zweckmäßig zu programmieren. Wir werden einige dieser Möglichkeiten beschreiben, bevor wir uns gute Praktiken der Ein/Ausgabe-Programmierung ansehen.

MÖGLICHKEITEN DES PRINT-BEFEHLS

PRINT-BEFEHL UND SEMIKOLON

Üblicherweise beendet ein PRINT-Befehl seine Darstellung mit einem RETURN-Befehl. Dadurch wird der nächste PRINT-Befehl veranlaßt, seine Darstellung am Anfang der folgenden Zeile zu beginnen. Das folgende Programm wird daher eine Spalte von 20 Zeichen auf der ersten Zeilenposition zur Darstellung bringen:

```
C$="W":FOR I=1 TO 20:? C$:NEXT:?"PHEW!"
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
PHEW!

READY.
▒
```

Ein Semikolon nach jeder Variablen eines PRINT-Befehls bewirkt, daß die Darstellung auf der nächsten verfügbaren Position erfolgt. Wird die letzte (oder einzige) Variable in der Parameterliste eines PRINT-Befehls mit einem Semikolon abgeschlossen, dann wird RETURN unterdrückt. Das folgende Programm stellt daher 800 Zeichen auf 20 Zeilen eines 40spaltigen Bildschirms dar:

```
C$="W":FOR I=1 TO 800:? C$;:NEXT:?"PHEW!"
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
PHEW!

READY.
```

Der Index I=1 TO 800 in der Programmschleife FOR-NEXT zählt die anzuzeigenden W's, in diesem Fall 800 Zeichen. Beim ersten PRINT-Befehl wird eine neue Zeile begonnen und das erste W dargestellt. Das Semikolon hinter der Variablen C$; verhindert ein RETURN (d.h. den Sprung auf den Anfang der folgenden Zeile), so daß der Kursor hinter dem ersten W liegenbleibt. Dort wird das zweite W dargestellt und der Kursor springt eine Position weiter. Dieses Spiel geht bis zum Zeilenende. Beim Darstellen des letzten W springt der Kursor auf die nächste Zeile in die erste Position, wo die Darstellung fortgesetzt wird.

Warum aber wird PFUI! auf einer neuen Zeile dargestellt? Es scheint eine neue Zeile zu beginnen, da das letzte W auf der letzten Position der vorangegangenen Zeile stand. Ändern Sie die Zeichenzahl von 800 auf 780, und PFUI! wird am Ende der Zeichenzeile dargestellt, wie das folgende Programm zeigt:

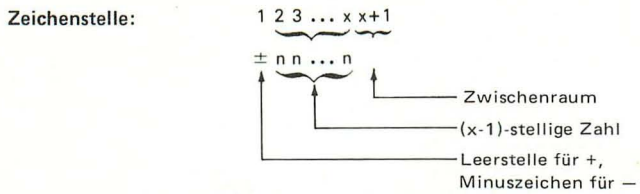
```
C$="-":FOR I=1 TO 780:?C$;:NEXT:?"PHEW!"
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------------------------------
------------------PHEW!
READY.
```

Das Semikolon verkettet Zeichen, indem es sie ohne Zwischenraum nebeneinander darstellt. Numerische Daten werden ebenfalls in einem fortlaufenden Zeilenformat dargestellt, wobei jedoch zwei Zwischenräume eingefügt werden: ein Zwischenraum zur Trennung der Zahlen, der zweite Zwischenraum zur Darstellung des Minuszeichens (das Pluszeichen wird durch eine Leerstelle dargestellt).

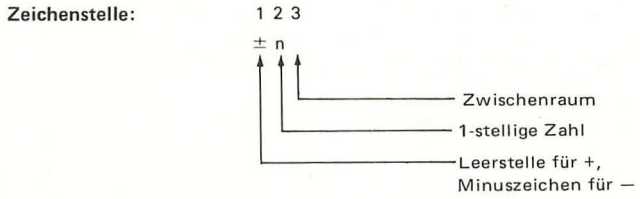
Verändern Sie zur Demonstration im letzten Programm das darzustellende Zeichen von einem Buchstaben in eine einstellige Zahl. Drei Zeichenstellen werden belegt, um jede Zahl darzustellen. Verändern Sie daher den Endwert des Index I von 800 auf 800/3=267. Die Zahl 5 würde dann wie folgt dargestellt werden:

```
C=+5:FOR I=1 TO 267:?C;:NEXT:?"PHEW!"
 5  5  5  5  5  5  5  5  5  5  5  5  5  5
5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5
5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5
5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5
5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5
5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5
5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5
5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 PHEW!

READY.
```

Beachten Sie den Zwischenraum von der letzten dargestellten Zahl bis zum Wort PFUI!. Das hat seinen einfachen Grund im Format der Zahlendarstellung:



Angewandt auf die einstellige Zahl 5 in obigem Beispiel:



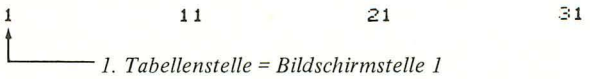
Mehrstellige Zahlen würden in unserem Programmbeispiel den Bildschirm sprengen, genauer: von der gesamten erforderlichen Zeilenzahl kann der Bildschirm nur 25 Zeilen darstellen, überzählige Zeilen werden über den oberen Bildschirmrand hinausgeschoben und verschwinden vom Bildschirm. Brauchten wir zur Darstellung eines W als Indexgrenze I=800, so brauchen wir zur Darstellung der Zahl 2001 ein sechsstelliges Darstellungsfeld, also als Indexgrenze I=800/6=134:

```
C=2001:FOR I=1 TO 134:?C;:NEXT:?"PHEW!"
 2001  2001  2001  2001  2001  2001  200
1  2001  2001  2001  2001  2001  2001  2
001  2001  2001  2001  2001  2001  2001
 2001  2001  2001  2001  2001  2001  200
1  2001  2001  2001  2001  2001  2001  2
001  2001  2001  2001  2001  2001  2001
 2001  2001  2001  2001  2001  2001  200
1  2001  2001  2001  2001  2001  2001  2
001  2001  2001  2001  2001  2001  2001
 2001  2001  2001  2001  2001  2001  200
1  2001  2001  2001  2001  2001  2001  2
001  2001  2001  2001  2001  2001  2001
 2001  2001  2001  2001  2001  2001  200
1  2001  2001  2001  2001  2001  2001  2
001  2001  2001  2001  2001  2001  2001
 2001  2001  2001  2001  2001  2001  200
1  2001  2001  2001  2001  2001  2001  2
001  2001  2001  2001  2001  2001  2001
 2001  2001  2001  2001  2001  2001  200
1  2001  2001  2001  2001  2001  2001  2
001 PHEW!

READY.
```

Zahlen werden durch ein Zeilenende geteilt, da das Semikolon (;) eine fortlaufende Darstellung bewirkt und lediglich ein Zeilenende den Sprung auf die folgende Zeile auslöst.

PRINT-BEFEHL UND KOMMA

Ein Komma nach einer darzustellenden Variablen oder am Ende einer PRINT-Anweisung führen zu einer Bildschirmdarstellung in Tabellenform mit einem Spaltenabstand entsprechend zehn Zeichen. Auf einem 40spaltigen Bildschirm erscheinen die Spalten in folgenden Positionen:



Ändern Sie in unserem obigen Programm das Semikolon in ein Komma. Auf einem 40spaltigen Bildschirm werden dadurch Zahlen in vier Spalten dargestellt. Bei vier Zahlen je Zeile und 20 Zeilen muß das Indexende auf I=4*20=80 gesetzt werden. Beachten Sie bei Ablauf des Programms, daß die erste Position einer jeden Spalte für das Vorzeichen reserviert ist (eine Leerstelle für ein Pluszeichen):

```
C=2001:FOR I=1 TO 80:?C,:NEXT:?"PHEW!"
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
 2001          2001          2001          2001
PHEW!

READY.
```

Die Verwendung von Kommas zwischen den Variablen eines PRINT-Befehls ist ein bequemer Weg zur Tabellierung. Und nun ändern Sie den Zahlenwert von C von 2001 auf 44, und das Indexende von 134 auf 200. Drücken Sie RETURN und (Überraschung!) folgendes Bild entsteht:

```
C=44:FOR I=1 TO 200:?C;:NEXT:?"PHEW!"
 440 442 44  440 442 44  440 442 44  440
 442 44  440 442 44  440 442 44  440 442
 44  440 442 44  440 442 44  440 442 44
 440 442 44  440 442 44  440 442 44  440
 442 44  440 442 44  440 442 44  440 442
 44  440 442 44  440 442 44  440 442 44
 440 442 44  440 442 44  440 442 44  440
 442 44  440 442 44  440 442 44  440 442
 44  440 442 44  440 442 44  440 442 44
 440 442 44  440 442 44  440 442 44  440
 442 44  440 442 44  440 442 44  440 442
 44  440 442 44  440 442 44  440 442 44
 440 442 44  440 442 44  440 442 44  440
 442 44  440 442 44  440 442 44  440 442
 44  440 442 44  440 442 44  440 442 44
 440 442 44  440 442 44  440 442 44  440
 442 44  440 442 44  440 442 44  440 442
 44  440 442 44  440 442 44  440 442 44
 440 442 44  440 442 44  440 442 44  440
 442 44  440 442 44  440 442 44  440 442
PHEW!HEW!

READY.
```

Einige der Ziffern der vorangegangenen Darstellung von 2001 wurden nicht gelöscht. CBM BASIC verwendet einen Sprungbefehl (CURSOR RIGHT) zwischen den Zeichenfeldern, und keinen Löschbefehl. Wenn Sie eine bestehende Bildschirmdarstellung überschreiben wollen, dann werden alte Zeichen zwischen den neuen Spalten nicht gelöscht. Beachten Sie ebenfalls das übriggebliebene PFUI! (siehe das Programmbeispiel zur zeilenweisen Darstellung der Zahl 2001); CBM BASIC stellt zwar das PFUI! dar, löscht aber nicht den Rest der Zeile. Das kann von großem Nutzen sein, wenn Sie an eine Bildschirmdarstellung etwas anfügen wollen, und Sie sollten diese Möglichkeit in Erinnerung behalten. Im vorliegenden Fall jedoch läßt das Programm störende Zeichen auf dem Bildschirm.

Um störende Zeichen vom Bildschirm zu entfernen, können Sie das Programm den Bildschirm löschen lassen, ehe eine neue Ausgabe beginnt. Zu diesem Zweck setzen Sie eine PRINT CLEAR SCREEN-Anweisung vor die Programmschleife FOR – NEXT:

```
C=44:?"■" FOR I=1 TO 200:?C$;:NEXT:?"PHEW!"
```

Bildschirm löschen (CLEAR-Taste)

Wenn Sie jetzt die RETURN-Taste drücken, werden Sie den Bildschirm leer sehen. Die Zahlendarstellung wird auf der zweiten Zeile beginnen.

Um die Darstellung auf der ersten Zeile beginnen zu lassen, fügen Sie ein Semikolon nach der PRINT CLEAR SCREEN-Anweisung ein:

```
C$="■":?"■";:FOR I=1T0840:?C$;:NEXT:?"PHEW!"
```

Kommas sind auch wirksam beim Ausgeben von Zeichenketten. Geben Sie als Beispiel folgendes Programm zur tabellierten Darstellung von Zeichen auf 20 Zeilen ein:

```
A$="EINS!":B$="ZWEI!":C$="DREI!":D$="VIER!"
FOR I=1 TO 20:?A$,B$,C$,D$:NEXT:?"PFUI!"
```

BEWEGUNG DES KURSORS

In Kapitel 3 diskutierten wir die Möglichkeiten der Bildschirmedition, die mit den folgenden Kontrolltasten des Kursors möglich sind: CLEAR SCREEN/HOME, CURSOR UP/DOWN, CURSOR LEFT/RIGHT, INSERT/DELETE und RETURN.

Die Tasteneingaben CLEAR SCREEN/HOME, CURSOR UP/DOWN, CURSOR LEFT/RIGHT und REVERSE können im Zusammenhang mit dem PRINT-Befehl verwendet werden. Nicht verwendet werden können jedoch die Tasteneingaben INSERT/DELETE und RETURN in einem PRINT-Befehl.

Die Kontrolltasten des Kursors werden in der Zeichenkette eines PRINT-Befehls als graphische Symbole dargestellt, bis der PRINT-Befehl ausgeführt wird:

```
100 PRINT"*■■■"
```

Rückkehr zum direkten Dialog
Symbol für Kursor nach rechts
Umschalten in Programm-Mode

Wird der PRINT-Befehl ausgeführt, dann können Sie an der Stellung der Sternsymbole erkennen, daß der Kursor nach rechts bewegt wurde:

```
RUN

*  *
```

Um einfache programmierte Kursorbewegungen zu üben, geben Sie folgendes Programm ein:

```
10  PRINT"<CLEAR SCREEN>";
20  PRINT"<CURSOR|>* <CURSOR|>* <CURSOR|>* <REVERSE><CURSOR|>*
    <CURSOR|>* <CURSOR|>*"
30  PRINT"<CURSOR|><CURSOR|><CURSOR|><CURSOR|>";
```

Auf dem Bildschirm sollte dieses Programm folgendes Aussehen haben:

```
10 PRINT"□";
20 PRINT"■*■*■*□*□*□*";
30 PRINT"■■■■"
40 END
```

Nach Programmausführung sollte folgendes Bild sichtbar sein:

```
       ■
 *    ■
  *  ■
    *
```

Das mag nicht das sein, was Sie erwartet haben. Wenn Sie erwartet haben, daß die Zeichenfolge:

```
20 PRINT "■*■*■*          "
```

die Sternsymbole in vertikaler Anordnung ausgibt:

```
*

*

*
```

oder daß der Befehl:

```
20 PRINT "          □*□*□*"
```

Sternsymbole über die drei bereits ausgegebenen Symbole schreibt,

```
■

■

■
```

dann haben Sie die automatische Rechtsbewegung des Kursors nach jeder Tasteneingabe vergessen. Die programmierte Kursorbewegung veranlaßt den CBM-Computer, den Kursor direkt auf oder ab zu bewegen, mit der Darstellung der Sternsymbole ist jedoch automatisch eine Fortschaltung des Kursors um jeweils eine Position nach rechts verbunden. Dadurch wird vermieden, daß das zuvor geschriebene Symbol vom nächsten Symbol überschrieben wird. Das folgende Diagramm zeigt die Kursorbewegungen in obigem Programm:

automatische Kursorbewegung

Um Zeichen in vertikaler Anordnung auszugeben, müssen Sie die automatischen Kursorbewegungen kompensieren, indem Sie den Kursor um eine Stelle nach links führen, ehe er nach unten oder oben bewegt wird. Das folgende Programm bewirkt die Darstellung von Sternsymbolen in vertikaler, absteigender Folge und daneben in vertikaler aufsteigender Folge:

```
20 PRINT "<CURSOR↓>*<CURSOR←><CURSOR↓>*<CURSOR←><CURSOR↓>*
   <REVERSE>*<CURSOR←><CURSOR↑>*<CURSOR←><CURSOR↑>*";
```

Die Programmdarstellung hat folgendes Aussehen auf dem Bildschirm:

```
20 PRINT"*■*■*■*■*■*■*";
```

Wenn Sie versuchen, die Tasten INSERT/DELETE und RETURN zu programmieren, werden Sie auf ein überraschendes Ergebnis stoßen.

Die INSERT-Taste ist nicht programmierbar, obwohl es so scheint. Wenn Sie versuchen, die INSERT-Taste zu programmieren, dann erscheint ein " ▌ " zwischen den beiden Anführungszeichen. Bei der Programmausführung ignoriert der CBM-Computer dieses INSERT-Symbol.

Die Tasteneingabe DELETE ist nur in direktem Dialog verwendbar. Die Verwendung der DELETE-Taste in einem PRINT-Befehl löscht einfach das vorangegangene Symbol, es sei denn, daß die DELETE-Taste innerhalb einer Folge von Zeicheneinfügungen mittels INSERT auftritt. Die DELETE-Taste ist programmierbar nach einem INSERT, machen Sie aber davon keinen Gebrauch. Es gibt Schwierigkeiten, und Sie können das gleiche Ziel auf einfachere Weise erreichen.

Die Tasteneingabe RETURN in einer PRINT-Anweisung bewegt den Kursor aus der Anweisung heraus auf den Anfang der folgenden Zeile.

ZEICHENEINGABE IM ASCII-KODE

Wenn Sie ein Symbol nicht mittels Tastendruck eingeben können, um eine Zeichenkette zu bilden, dann kann das Symbol auch mittels seines ASCII-Kodes aufgerufen werden.

Die Funktion CHR$ übersetzt einen ASCII-Kode in seine Symboldarstellung. Das Format von CHR$ ist folgendes:

```
PRINT CHR$(xx)
```

Zahl zwischen 0 und 255
(siehe Anhang A, Tabelle A-4, ASCII)

Den richtigen ASCII-Kode für das gewünschte Symbol entnehmen Sie Anhang A. Durchlaufen Sie die Spalten der Tafel, bis Sie ein gewünschtes Symbol oder Kursorbewegung gefunden haben, und notieren Sie die zugehörige Kodenummer in ASCII. Setzen Sie diese Zahl zwischen die Klammern der Funktion CHR$. Für das Symbol $ finden Sie zum Beispiel zwei Zahlen: 36 und 100. Beide Zahlen bewirken dasselbe. Für guten Programmierstil sollte man sich im gesamten Programm für eine der beiden Zahlen entscheiden. Wir werden 36 verwenden und wie folgt in die CHR$-Funktion einsetzen:

```
PRINT CHR$(36)
```

Versuchen Sie, das Symbol auszugeben:

```
PRINT CHR$(36)
$
```

Und nun versuchen Sie den gleichwertigen ASCII-Kode 100:

```
PRINT CHR$(100)
$
```

Das Ergebnis ist das gleiche. Machen Sie weitere Versuche mit den übrigen ASCII-Kodes von 0 bis 255.

Die Funktion CHR$ läßt sich wie folgt in Verbindung mit einem PRINT-Befehl verwenden:

```
10 PRINT CHR$(36);CHR$(42);CHR$(166)

RUN
$*▒
```

Die Funktion CHR$ gestattet die Aufnahme von im übrigen nicht verfügbaren Symbolen in die Zeichenkette einer PRINT-Anweisung, so zum Beispiel die Kontrollzeichen für RETURN, INSERT/DELETE und das Anführungszeichen. Ebenso kann die Funktion CHR$ zur Prüfung des Vorhandenseins von Kursoreingaben wie RETURN und INSERT/DELETE durch Vergleich verwendet werden. Nehmen Sie an, ein Programm soll die Zeicheneingaben am Tastenfeld auf das Auftreten einer RETURN-Eingabe überprüfen. Sie könnten das Vorhandensein einer RETURN-Eingabe (mit dem ASCII-Kode 13) wie folgt feststellen:

```
10 GET X$:IF X$<>CHR$(13) THEN 10
```

Diese Prüfung wäre nicht möglich, wenn Sie ein RETURN zwischen Anführungszeichen verwenden würden:

```
20 IF X$<>"[RETURN]"THEN 10
```

nicht möglich

Würden Sie die RETURN-Taste nach Anführungszeichen drücken, dann bewegte sich der Kursor automatisch auf den Anfang der folgenden Zeile:

```
20 IF X$<>"
```

Taste RETURN gedrückt

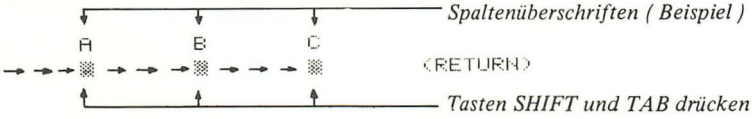
EINGABEMÖGLICHKEITEN AN CBM 8032-COMPUTERN

KURSORBEWEGUNGEN

Der Bildschirm-Editor in Version 4.1 für CBM 8032-Computer verfügt über zwei neue Funktionen und einige neue Editions- und Kontrollmöglichkeiten. Die neuen Funktionen werden durch die Tasten TAB und ESCAPE dargestellt. Die neuen Editions- und Kontrollmöglichkeiten beinhalten eine programmierbare Glocke, Zeileneinfügung und Zeilenlöschung, Löschung des Bildschirms, Umschaltung zwischen graphischen Symbolen und Text sowie fortlaufende Textdarstellung in einem Bildschirmfenster programmierbarer Größe.

Die Tabellierfunktion und die TAB-Taste

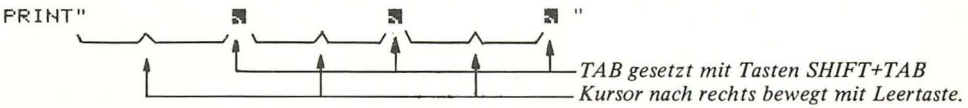
Die TAB-Taste arbeitet ähnlich wie die TAB-Taste einer Schreibmaschine. Bis zu 80 Spalten können je Zeile markiert werden. Um in direktem Dialog mit dem Computer eine Tabelle anzulegen, bewegen Sie den Kursor zur gewünschten Spalte und drücken dann gleichzeitig die Tasten SHIFT und TAB. Nach dem Markieren aller Spalten drücken Sie die RETURN-Taste:



Spaltenüberschriften (Beispiel)

Tasten SHIFT und TAB drücken

Sie können aber auch das Tabellenanlegen mit einer PRINT-Anweisung programmieren. Bewegen Sie den Kursor mit CURSOR RIGHT-Befehlen in der PRINT-Anweisung zu den gewünschten Tabellenspalten und markieren Sie die Spaltenposition mit TAB SET, d.h. durch die Tasten SHIFT und TAB. Sehen Sie das folgende Beispiel:



TAB gesetzt mit Tasten SHIFT+TAB

Kursor nach rechts bewegt mit Leertaste.

Vor jedem Tabelleneintrag drücken Sie zuerst die Taste TAB ,wodurch der Kursor zur nächsten Tabellenspalte springt und auf Ihre Eingabe wartet.

TAB SET wird in einem PRINT - Befehl mit dem Symbol ▜ dargestellt.

TAB SET kann auch mit dem ASCII-Kode 137 erzeugt werden, d.h. TAB SET kann mit Hilfe der Funktion CHRS(137)programmiert werden.

```
PRINT"▮▮▮▮▮";CHR$(137)
```

Die Taste TAB läßt den Kursor zur nächsten Tabellenspalte auf dem Bildschirm wandern. Wird TAB gedrückt, wenn der Kursor auf der letzten Tabellenspalte steht, dann wandert der Kursor bis zum Zeilenende.

Die Tabellierung erfolgt dort, wo in der PRINT-Anweisung das Symbol für die TAB-Taste steht. Hier ein Beispiel:

```
PRINT"UEBER▜SCHRIFT          UEBER▜SCHRIFT          UEBER▜SCHRIFT"
```

Tab gesetzt

```
UEBERSCHRIFT          UEBERSCHRIFT          UEBERSCHRIFT
```

resultierende Spaltenpositionen

Eine mit TAB SET gesetzte Spaltenposition kann mit TAB CLEAR gelöscht werden. TAB CLEAR entsteht durch gleichzeitiges Drücken der Tasten SHIFT und TAB Bewegen Sie den Kursor zu der Stelle, an der eine Spaltenposition gelöscht werden soll, und drücken Sie SHIFT und TAB. Nach dem letzten TAB CLEAR drücken Sie RETURN.

TAB CLEAR und TAB SET werden auf gleiche Weise mittels der Tasten TAB und SHIFT gebildet. Wenn Sie also versuchen, dort eine Tabelle zu löschen, wo keine war, werden Sie stattdessen eine Tabelle einrichten.

Zum programmierten Löschen einer Tabelle bewegen Sie den Kursor mit Hilfe einer PRINT-Anweisung zur gewünschten Spaltenposition und bilden das TAB CLEAR-Symbol aus SHIFT und TAB - Taste:

```
20 PRINT"▮▮▮▮▮▮▮▮▮▮▜▮▮▮▮▮▮▮▮▮▮▜▮▮▮▮▮▮▮▮▮▜ "
```

TAB gelöscht

Kursor nach rechts

TAB CLEAR wird wie TAB SET mit dem Symbol ▜ dargestellt.

TAB CLEAR kann aber auch mit seiner ASCII-Kodenummer und der Funktion CHR$ programmiert werden:

```
PRINT"▮▮▮▮▮";CHR$(137)
```

CHR$(137) stellt beides dar: TAB SET und TAB CLEAR.

Die ESCAPE-Taste

Auf Computern der Serie CBM 2001 erzeugt die ESCAPE-Taste einen ASCII-Kode, hat aber keine Funktion bei der Textedition. Auf den CBM 8032-Computern besitzt die ESCAPE-Taste zwei Funktionen: in direktem Dialog mit dem Computer verwendet, löscht sie die Tastenanschläge INSERT, REVERSE oder eine Texteingabe. ESCAPE gestattet außerdem, gewisse Zeichenketten als Kontrollfunktionen für eine Bildschirmedition zu interpretieren.

ESCAPE kann mit Hilfe der Funktion CHR$ in eine PRINT-Anweisung aufgenommen werden (Zahlenwert des ASCII-Kodes für ESCAPE ist 27 nach Tabelle A-2):

```
PRINT CHR$(27)
```

KONTROLLFUNKTIONEN

Die nachfolgend aufgeführten Kontrollfunktionen sind nur auf Computern der Serie CBM 8001 mit 80spaltigem Bildschirm verfügbar. Eine ausführliche Darstellung dieser Funktionen findet sich in Kapitel 8. Einige Anwendungsbeispiele werden am Ende dieses Kapitels gegeben.

Alle diese Funktionen wurden entwickelt, um die Bildschirmdarstellung und die Dateneingabe zu verbessern; obwohl sie in direktem Dialog mit dem Computer verwendet werden können, sollten sie nicht zur Edition von Programmen eingesetzt werden, da viele von ihnen zwar das Schirmbild, nicht jedoch Speicherinhalte im Computer verändern.

Beim Einsatz einer dieser Kontrollfunktionen muß ihre Symboldarstellung in die Parameterliste einer PRINT-Anweisung aufgenommen werden. Der Funktionscharakter einer Symboldarstellung wird innerhalb einer Zeichenkette durch ein Kontrollzeichen, und außerhalb einer Zeichenkette durch die Funktion CHR$ angezeigt. Das Kontrollzeichen wird durch Drücken der ESCAPE-Taste gebildet, anschließend durch Drücken der REVERSE-Taste, und schließlich durch Eingabe der entsprechenden Buchstaben ohne Bedienung von SHIFT.

Glocke. Die Glockenfunktion arbeitet nur auf CBM-Computern, die mit einer Glocke ausgerüstet sind. Die Glocke schlägt automatisch an, wenn der Computer in Betrieb genommen wird und wenn der Kursor über Spalte 75 läuft. Wird mit einem verkleinerten Bildschirmfenster gearbeitet (siehe unten), dann ertönt die Glocke fünf Zeichen vor Ende der verkürzten Zeile. Die Glocke kann außerdem mit der Anweisung PRINT CHR$(7) zum Ertönen gebracht werden.

Zeilenlöschung bzw. Zeileneinfügung. Diese Funktionen löschen eine Bildschirmzeile bzw. fügen eine Zeile ein. Die DELETE LINE-Funktion löscht die Zeile, auf der der Kursor steht, und läßt die darunter befindlichen Zeilen um eine Zeile nach oben rutschen. Die INSERT LINE-Funktion fügt dort eine Zeile ein, wo der Kursor steht, indem die Kursorzeile und alle darunter befindlichen Zeilen um eine Stelle nach unten verschoben werden. Die unterste Zeile wird hierbei aus dem Bildschirm geschoben. Weder die DELETE LINE- noch die INSERT LINE-Funktion werden vom Speicher des Computers registriert, lediglich das Schirmbild ändert sich. Die Funktion DELETE LINE wird mit der Funktion CHR$(21) in der Parameterliste einer PRINT-Anweisung ausgedrückt. Die Funktion INSERT LINE wird über CHR$(149) in der PRINT-Anweisung ausgedrückt.

Löschung bis Zeilenende/-anfang. Diese Funktionen löschen Teile derjenigen Zeile, auf der sich der Kursor befindet. Die Funktion ERASE BEGIN löscht den Text links des Kursors, die Funktion ERASE END löscht den Text rechts des Kursors. Keine der Funktionen ist verbunden mit einer Textverschiebung. Keine der Funktionen verändert den Speicherinhalt des Computers. In der Parameterliste einer PRINT-Anweisung werden ERASE END mit CHR$(150) und ERASE BEGIN mit CHR$(22) gebildet.

Umschaltung Graphik/Text. Die Funktion GRAPHIC wählt graphische Symbole aus dem Standardsatz aus und stellt sie ohne Zwischenraum dar, so daß die Qualität einer Graphik verbessert wird. Die Funktion TEXT bringt Buchstaben in Groß- und Kleinschreibung zur Darstellung. In der Parameterliste einer PRINT-Anweisung wird die Funktion GRAPHIC mit CHR$(142) angesprochen, während die Funktion TEXT mit CHR$(14) aufgerufen wird. Die TEXT-Funktion erlaubt, den alternativen Satz an graphischen Zeichen anzusprechen, während weiterhin Groß- und Kleinbuchstaben gewählt werden können.

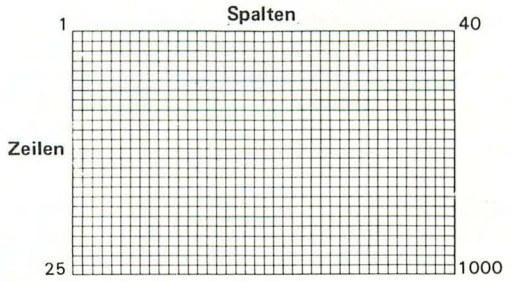
Programmierbares Bildschirmfenster. Es gibt vier Funktionen, mit denen auf CBM-Computern der Serie 8000 ein Fenster für die Bildschirmdarstellung definiert werden kann, wobei ein Text hinter diesem Fenster entweder abwärts oder aufwärts durchlaufen kann. Die SET TOP-Funktion nimmt die augenblickliche Kursorstellung als obere linke Ecke des Fensters, während die SET BOTTOM-Funktion eine augenblickliche Stellung des Kursors als rechte untere Ecke des Fensters versteht. Dieses Fenster kann jederzeit durch zweimaliges Drücken der Taste HOME gelöscht werden, oder durch zwei aufeinanderfolgende HOME-Angaben in der Parameterliste einer PRINT-Anweisung. SET TOP wird durch CHR$(15) und SET BOTTOM durch CHR$(143) dargestellt; diese CHR$-Funktionen sollten in der Parameterliste einer PRINT-Anweisung denjenigen Symbolen folgen, mit denen der Kursor zu den Fensterecken bewegt wurde.

Durchlaufender Bildschirmtext. Die Funktion SCROLL UP bewegt Text, der im oben definierten Bildschirmfenster sichtbar ist, um eine Zeile nach oben. Eine Leerzeile wird von unten in den Text eingefügt. Ganz ähnlich die Funktion SROLL DOWN, mit der Text um eine Zeile nach unten bewegt werden kann, wobei eine Leerzeile von oben dem Text zugefügt wird. Beide Funktionen können über die Parameterliste eines PRINT-Befehls aufgerufen werden, wobei SCROLL UP als CHR$(25) und SCROLL DOWN als CHR$(153) in der Parameterliste dargestellt werden.

ZEICHENAUSGABE AUF KALKULIERTE RASTERPUNKTE DES BILDSCHIRMS

Mit dem Befehl POKE können Sie jedes Symbol an jeder Stelle des Bildschirms zur Darstellung bringen. Sie bringen einfach mittels POKE das Symbol in diejenige Speicherstelle, die zur gewünschten Stelle auf dem Bildschirm gehört.

Der Bildschirm eines CBM-Computers ist wie ein Rasternetz aufgebaut, das aus 25 Zeilen und 40 (oder 80) Spalten besteht und 1000 (oder 2000) Karos bildet. Das folgende Bild zeigt den Aufbau eines 40spaltigen Bildschirms:



In jedem Karo kann ein Symbol dargestellt werden. Jedem Karo entspricht eine Speicherstelle und eine Speicheradresse im Speicher des Computers. Der Bildschirmspeicher beginnt bei Adresse 32768 für das erste Karo (Zeile 1, Spalte 1) und endet mit Adresse 33767 für Karo 1000 (Zeile 25, Spalte 40) bzw. mit Adresse 34767 für Karo 2000 (Zeile 25, Spalte 80). Die Speicheradresse 32768 entspricht also dem Raster-

punkt (1,1), Speicheradresse 32769 entspricht dem Rasterpunkt (1,2) usw., wobei die Koordinatenangabe (1,2) Zeile 1, Spalte 2 bedeutet.

Die unten stehende Figur zeigt den Zusammenhang zwischen Karo auf dem Bildschirm und entsprechender Adresse im Speicher. Um die Adresse für jedes Karo auf dem Bildschirm zu finden, können Sie folgende Gleichungen zur Berechnung benutzen:

40 - spaltiger Schirm

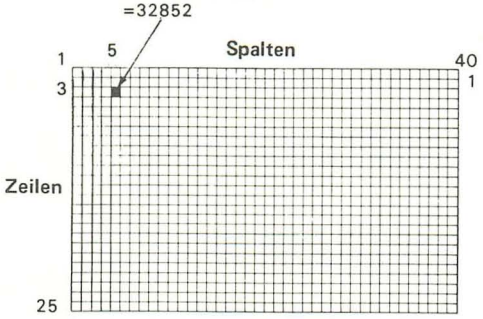
32768+(Spalte-1)+(40*(Zeile -1))

80- spaltiger Schirm

32768+(Spalte-1)+(80*(Zeile -1))

Geben Sie die Zeilen- und Spaltenzahl jedes Karos in die Gleichung ein, und Sie erhalten seine Speicheradresse. Für den Rasterpunkt (3,5), d.h. das Karo in Zeile 3, Spalte 5, ergibt sich die Speicheradresse 32852, wie folgende Rechnung zeigt:

```
=32768+ (Spalte-1)+(40*(Zeile-1))
=32768+(5-1)+(40* 3-1)
=32768+4+(40*2)
=32768+4+80
=32852
```



Diese Gleichungen machen es möglich, Symbole auf dem Bildschirm an berechenbaren Rasterpunkten darzustellen, wenn man die Zeile und die Spalte kennt, an die der POKE-Befehl das Symbol bringen soll. Der POKE-Befehl hat folgendes Format:

```
POKE A,X
```

wobei: A **Adresse im Bildschirmspeicher**
X **darzustellendes Symbol**

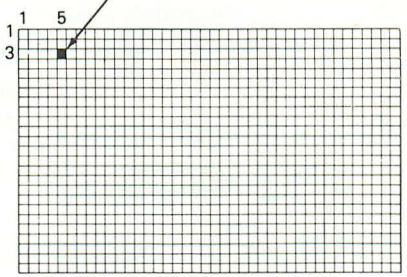
Wenn Sie jetzt A durch die Gleichung zur Adressenberechnung ersetzen, wird der Computer Ihnen die Speicheradresse des Rasterpunkts ausrechnen:

```
POKE 32768+ (Spalte -1)+(40*(Zeile-1)), X
POKE                A                   X
```

Ein Beispiel: Werden Zeile (=Z) und Spalte (=S) mit den Werten 3 und 5 eingegeben, und als Symbol ein $, dann wird ein Dollarzeichen auf dem Rasterpunkt (3,5) mit der Adresse 32852 dargestellt:

```
10 INPUT C,R,X
20 POKE 32768+(C-1)+(40*(R-1)),X
```

=32768+ (Spalte-1)+(40*(zeile-1)), X
=32768+(5-1)+ (40*(Zeile-1)), X
=32768+4+(40*2),X
=32768+4+80,X
=32852,X



Das für X am Tastenfeld eingegebene Symbol wird in Form einer Zahl abgespeichert. Das der Zahl im ASCII-Kode entsprechende Symbol wird beim Lesen des Speichers zur Darstellung gebracht.

Im POKE-Befehl können Variable verwendet werden, doch muß die Variable schließlich einen Zahlenwert innerhalb erlaubter Grenzen annehmen:

POKE 32768+A,X

wobei: A eine Zahl zwischen 0 und 999

POKE A, X

wobei: A eine Zahl zwischen 32768 und 33767

für einen 40-spaltigen Bildschirm:
32768+999=33767

Die Verwendung einer Variablen zur Darstellung der Adresse eines Rasterpunkts ist ratsam bei einer Folge von Bildschirm-Ausgaben. Das folgende Programmbeispiel stellt ein Symbol X mit Zwischenräumen von 10 Leerstellen über den gesamten Bildschirm dar:

```
10 A=32768
20 POKE A,X
30 A=A+10
40 IF A<=33767 GOTO 20
```

AUFBAU GUTER PROGRAMME ZUM EINLESEN VON DATEN

Ein Einleseprogramm sollte Daten blockweise entsprechend ihrer funktionellen Zusammengehörigkeit einlesen.

Ein Versandlistenprogramm zum Beispiel erfordert das Einlesen von Namen und Adressen als Daten. Sie sollten jeden Namen mit Adresse als eine funktionelle Einheit behandeln. Mit anderen Worten: Ihr Programm sollte nach Name und Adresse

fragen, so daß der Programmbenutzer all diese Informationen eingeben kann und dann Gelegenheit für Korrekturen hat; ist der Benutzer überzeugt, daß Name und Adresse richtig sind, dann sollte das Programm Name und Adresse als funktionelle Einheit bearbeiten und anschließend nach dem nächsten Namen mit Adresse fragen.

Schlechte Programmierpraxis wäre es, Daten in ihre kleinsten Bestandteile aufzuteilen. Im Fall des Versandlistenprogramms wäre es schlechte Programmierpraxis, nach dem Namen zu fragen, diese Daten sofort nach Eingabe zu verarbeiten, und dann nach der Adresse zu fragen, wodurch Name und Adresse als funktionell verschiedene Einheiten behandelt werden.

Das Ziel jedes Programms zum Einlesen von Daten sollte sein, einem Benutzer die Fehlererkennung leicht zu machen und ihm möglichst viel Gelegenheit zur Fehlerkorrektur zu geben.

Nehmen Sie den Fall eines Programms, dem eine lange Liste kurzer, identischer Daten eingegeben werden muß. Solch eine Liste besteht etwa aus Namen, Personenkennzahlen und weiteren Daten. Ein gut konzipiertes Programm liest solche Daten blockweise ein. Als Beispiel das Einlesen von Namen: das Programm mag dem Benutzer erlauben, so viele Namen einzulesen, wie in eine Spalte des Bildschirms passen, so daß jede Eingabe korrigiert werden kann solange sie noch auf dem Bildschirm sichtbar ist. Das Programm könnte die Namen übernehmen und verarbeiten, sobald sie aus dem oberen Ende des Bildschirms herausgewandert sind. Die Alternative wäre ein Einleseprogramm, das jeweils nur einen Namen liest und verarbeitet. Ein solches Programm würde aber die Chance des Benutzers, Fehler zu erkennen und zu korrigieren, sehr einschränken.

Unter Umständen jedoch ist die blockweise Eingabe von Daten nicht der beste Weg, und überraschenderweise ergeben sich diese Umstände bei der Eingabe großer Datenmengen über das Tastenfeld. Nehmen Sie als Beispiel einen Benutzer, der hunderte von Namen und Adressen pro Tag über das Tastenfeld eingeben muß. Erfahrungen zeigen, daß die höchste Eingabeleistung richtiger Daten erreicht wird, wenn der Benutzer bei der ersten Eingabe alle gemachten Fehler ignorieren kann. Das Einleseprogramm solle keine Korrekturen zulassen, selbst wenn die Fehler schon beim Einlesen erkennbar sind. Der Benutzer sollte sich sogar üben, Eingabefehler zu ignorieren und stattdessen die Eingabe so schnell wie möglich auszuführen. Solche Datenmengen sollten doppelt eingegeben werden, vorzugsweise durch zwei Personen, um sie dann zu vergleichen. Die Wahrscheinlichkeit, daß zwei Personen genau denselben Fehler bei der Eingabe machen, ist derart gering, daß Sie annehmen können, alle Fehler gefunden zu haben, die beim Vergleich beider Dateneingaben gemeldet wurden. Ein weiteres Programm sollte die Korrektur der Eingabefehler ermöglichen.

INTERAKTIVE DATENEINGABE

Um den Wert guter, interaktiver (d.h. mit einem Rede-Antwort-Spiel zwischen Benutzer und Computer verbundener) Dateneingabe zu demonstrieren, werden wir mit einem sehr einfachen Beispiel beginnen. Beginnend mit dem Programm BLANKET werden wir Schritt für Schritt Programmänderungen diskutieren, die zur Verbesserung der Dateneingabe und der Benutzbarkeit des Programms führen.

Wir beginnen mit dem folgenden Programm:

```
100 C$="A"
110 PRINT "J"
120 FOR I=1 TO 840
130 PRINT C$;
140 NEXT
150 PRINT "PHEW!"
```

Dieses Programm stellt 840 Mal den Buchstaben A dar, abgeschlossen mit dem Ausruf PFUI!

Nehmen Sie an, wir wollen statt des A ein X dargestellt sehen.
Zuerst löschen wir den Zuweisungsbefehl im Programm. Zum Löschen einer Programmzeile geben Sie die Zeilennummer ein und drücken die RETURN-Taste.

```
LIST 100

 100 C$="A"

READY.
100◄──────────────── Eingabe der Zeilennummer, danach Taste RETURN drücken
LIST

 110 PRINT "■";
 120 FOR I=1 TO 840
 130 PRINT C$;
 140 NEXT
 150 PRINT "PHEW!"
READY.
```

Zeile 100 ist aus dem Programm verschwunden. Geben Sie jetzt den Befehl C$="X" ein und lassen das Programm ablaufen:

nach RETURN	**vor RETURN**
C$="X"	PHEW!
READY.	READY.
RUN	

Leerstellen und das Wort PFUI! wurden dargestellt, nicht jedoch der Buchstabe X. Offensichtlich wurde die Zuweisung zu C$ nicht dem Programm mitgeteilt.

Der Befehl RUN löscht alle Variablenzuweisungen und setzt sie auf 0, bevor das Programm gestartet wird. Daher wurde C$ auf Null gesetzt, und das Symbol Null wurde in der Programmschleife ausgegeben (eine Null bedeutet "Nichts": weder eine Ausgabe noch eine Kursorbewegung).

Gibt es einen Weg, der Variablen C$ in direktem Dialog mit dem Computer einen Wert zuzuweisen? Benutzen Sie statt RUN, das Variable auf einen Anfangswert setzt, den Befehl GOTO, für Programmzeile 110 als GOTO 110. Dadurch wird kein Wert einer Variablen verändert.

Die Prozedur zur Programmänderung ist daher folgende:

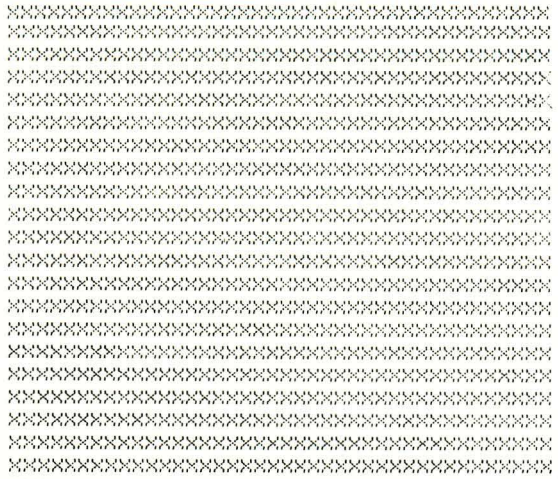
1. Geben Sie in direktem Dialog mit dem Computer die Zuweisung C$="Y" ein, wo Y ein beliebiges Symbol sein kann.
2. Geben Sie den Befehl GOTO 110 ein.

Es sind zwar nur zwei Schritte erforderlich, dennoch ist diese Prozedur umständlich. Sie müssen eine Zeile eingeben (die Zuweisungszeile), und wenn Sie den Befehl RUN statt GOTO eingeben, haben Sie die Prozedur von vorne zu beginnen.

vor RETURN

```
C$="X"

READY.
GOTO 110
```

nach RETURN



```
PHEW!

READY.
```

Das Programm könnte aber während seines Ablaufs ein am Bildschirm dargestelltes Symbol lesen, wenn Sie den GET-Befehl benutzen. Geben Sie folgende Zeile ein:

```
100 GET C$:IF C$="" GOTO 100
```

Lassen Sie sich das Programm auflisten, um sicher zu sein, die Zeile richtig eingegeben zu haben. Dann starten Sie das Programm. Drücken Sie irgendeine Symboltaste. Das eingegebene Symbol wird 840 Mal auf dem Bildschirm dargestellt. Starten Sie das Programm wieder. Drücken Sie die Taste eines anderen Symbols, und der Bildschirm zeigt das neue Symbol.

Hier die neue Prozedur zum Start des Programms BLANKET:

1. Geben Sie den RUN-Befehl ein.
2. Drücken Sie irgendeine Taste.

Das ist ein wirklicher Fortschritt gegenüber dem ursprünglichen Programm. Es ist jedoch ein bißchen verwirrend, daß der Bildschirm völlig gelöscht wird, während er auf Ihre Tasteneingabe wartet. Fügen Sie eine Aufforderung am Programmanfang ein, mit der nach einer Tasteneingabe gefragt wird. Geben Sie folgende Zeile ein:

```
90 ?"EINE TASTE DRUECKEN"
```

Lassen Sie sich das Programm zur Überprüfung ausgeben.

Jetzt gibt das Programm Benutzungshinweise. Lassen Sie das Programm mehrere Male ablaufen und bemerken Sie, wie leicht es jetzt zu benutzen ist.

Eine wichtige Änderung ist noch erforderlich. Wenn Sie das Programm mehr als einmal ablaufen lassen wollen, kehren Sie zum Programmanfang zurück, statt zum Programmende zu gehen. Sie müssen dann nicht den Befehl RUN eingeben, um das Programm wieder zu starten. Fügen Sie folgende Zeile ein:

```
160 GOTO 90
```

Nach erneuter Auflistung sollte das Programm wie folgt aussehen:

```
LIST

  90  PRINT "EINE TASTE DRUECKEN"
  100 GET C$:IF C$="" GOTO 100
  110 PRINT " ";
  120 FOR I=1 TO 840
  130 PRINT C$
  140 NEXT
  150 PRINT "PFUI!"
  160 GOTO 90
READY
```

Jetzt ist es noch einfacher zu benutzen. Geben Sie RUN ein und befolgen Sie die Anweisung.

Zur Beendigung der Programmbenutzung haben Sie natürlich die STOP-Taste zu bedienen. Auch das kann vermieden werden, indem eine bestimmte Taste programmiert wird, die Programmläufe zu beenden. Es könnte zum Beispiel die RETURN-Taste verwendet werden.

Lassen Sie uns sehen, wie das geht.

Alle Symboltasten und Kontrolltasten des Kursors verursachen Eingaben, die wie Zeichenketten auf ihr Vorhandensein abgefragt werden können. Der folgende Befehl prüft die Eingaben auf das Symbol "Y":

```
100 GET C$:IF C$="" GOTO 100
105 IF C$="Y" GOTO 200
```

Die RETURN-Taste stellt ein besonderes Problem dar. Sie können RETURN nicht wie eine Zeichenkette ansprechen:

" " ← *nicht verwendbar*

Das geht deshalb nicht, weil jede Eingabe von RETURN in CBM BASIC als Programmzeile gespeichert und danach auf den Anfang der nächsten Zeile gesprungen wird. Sie können aber die Funktion CHR$ benutzen, um nach der Eingabe von RETURN zu fragen. CHR$ erlaubt Ihnen, der Variablen einer Zeichenkette (hier: C$) den Zahlenwert eines ASCII-Kodes (hier: Zahlenwert 13 für RETURN) zuzuordnen und sie dann wie eine Zeichenkette zu behandeln.

Lassen Sie uns aber zuerst bedenken, was beim Vorliegen eines RETURN zu geschehen hat. Die letzte Zeile des Programms verzweigt zurück auf den Programmanfang. Um die Programmausführung zu beenden, müssen Sie unter die letzte Programmzeile springen. Fügen Sie hier folgende Zeile an:

```
170 END
```

Fügen Sie jetzt die Prüfung auf eine RETURN-Eingabe in Zeile 105 hinzu. Mit dieser Eingabe wird der Programmablauf beendet:

```
105 IF C$=CHR$(13) GOTO 170
```

Wir hätten die Zeilen 105 und 170 auch vereinigen können:

```
105 IF C$=CHR$(13) THEN END
```

Es ist jedoch gute Programmierpraxis, den Programmendebefehl an das physikalische Ende des Programms zu setzen, da es schwierig ist, Endebefehle inmitten eines Programms zu suchen.

Bei Unterdrückung der READY-Nachricht nach jedem Programmlauf wären zwei weitere Zeilen verfügbar, auf die 80 weitere Symbole ausgegeben werden könnten (bei einem 40spaltigen Bildschirm). Ändern Sie die Zahl der dargestellten Symbole von 840 auf 840+80=920 (in Zeile 120), so daß Zeile 120 wie folgt lautet:

```
120 FOR I=1 TO 920
```

Lassen Sie jetzt das Programm ablaufen, und Sie werden finden, daß es alle Zeilen um eine Stelle nach oben schiebt und eine Leerzeile an der Unterseite des Schirmbildes einfügt. Das kommt daher, daß CBM BASIC nach jeder Aufforderung EINE TASTE DRÜCKEN ein RETURN (Wagenrücklauf) ausführt, um die folgende Zeile für die erwartete Eingabe vorzubereiten. Wir können das sichtbar machen, indem wir den Kursor blinken lassen.

Üblicherweise können Sie den Kursor nicht sehen, da sein Blinken vor einem Programmlauf verhindert wird. Sie können den Kursor jedoch mit folgendem Befehl am Programmanfang zum Blinken bringen:

```
30 POKE 548,0
```
Kursor sichtbar machen

Das ist eine Speicherstelle, die in Kapitel 7 näher beschrieben wird.
Lassen Sie das Programm ablaufen, um zu sehen, ob der Kursor auf der untersten Zeile blinkt:

```
"
"
PFUI!
EINE TASTE DRUECKEN
```

Unser Programm hat den Kursor nicht nötig, löschen Sie daher Zeile 80.

Um die Leerzeile am unteren Ende des Bildschirms zu vermeiden, fügen Sie ein Semikolon zu dem PRINT-Befehl auf Zeile 90. Wir sollten außerdem eine Rückmeldung einfügen, aus der hervorgeht, daß RETURN zum Verlassen des Programms vorgesehen wurde. Mit diesen Änderungen sollte Zeile 90 folgendermaßen aussehen:

```
90 ?"EINE TASTE DRUECKEN ODER MIT RETURN BEENDEN";
```

Als letzte Aufgabe können Sie das Programm noch einmal durchlesen und Bemerkungen hinzufügen. Kommentieren Sie beispielsweise, wie die Zahl 920 entstanden ist; Sie können die Bemerkung auf die gleiche Zeile schreiben, indem Sie einen Doppelpunkt als Trennsymbol verwenden:

```
120 FOR I=1 TO 920    :REM 920/40= 23 ZEILEN
```

Fügen Sie zur Erinnerung hinzu, daß der Bildschirm gelöscht wird:

```
110 PRINT"□";          REM LOESCHEN DES BILDSCHIRMS
```

Schließlich fügen Sie noch am Programmanfang einige Zeilen zur Beschreibung des Programms hinzu. Figur 5-1 am Anfang von Kapitel 5 gibt die endgültige Version dieses Programms, genannt BLANKET, wieder. Sie können es auf Band oder Diskette speichern.

AUFFORDERUNG ZU BENUTZEREINGABEN

Jedes Programm, das eine Dateneingabe erfordert, sollte dem Benutzer Entscheidungsmöglichkeiten geben, indem es sich mit Fragen meldet. Fragen werden für gewöhnlich einzeilig dargestellt und erfordern eine einfache Antwort in Form eines "ja" oder "nein", eines Wortes oder einer Zahl. Beispielsweise kann die folgende Nachricht ausgegeben werden:

```
WUENSCHEN SIE AENDERUNGEN?
```

Ein Benutzer muß auf diese Nachricht mit der Eingabe eines JA oder NEIN antworten. Häufig genügen die Buchstaben J oder N. In einem anderen Fall mag der Benutzer eine Anzahl von Wahlmöglichkeiten haben. Die Nachricht:

```
WELCHE DER EINGABEN SOLL GEAENDERT WERDEN?
```

ermöglicht dem Benutzer, die zu ändernde Dateneingabe mit einer Zahl zu bezeichnen.

Programme für diese Art von Dialog sollten als unabhängige Unterprogramme geschrieben werden, die keine Kenntnis über das aufrufende Programm benötigen. Das zieht folgendes nach sich:

1. Sie können nicht sicher sein, daß die Nachricht auf eine leere Zeile ausgegeben wird. Wenn die Zeile nicht leer ist, wird die Nachricht alle vorhandenen Zeichen überschreiben; das Schlimmste aber ist, daß der verbleibende Textrest, der hinter der Nachricht stehen bleibt, als Teil der Antwort des Benutzers interpretiert wird.
2. Das Unterprogramm muß vom aufrufenden Programm Parameter erhalten. Fragt die Nachricht an den Benutzer zum Beispiel nach der Eingabe einer Zahl, dann sollte das aufrufende Programm den zulässigen Zahlenbereich an das Unterprogramm mitteilen.
3. Das Unterprogramm muß die Antwort des Benutzers an das aufrufende Programm zurückmelden. Diese Variable kann ein Buchstabe (z.B. J oder N), ein Wort (z.B. ja oder nein) oder eine Zahl sein.

Die Unterprogrammlogik kann nicht erkennen, auf welcher Zeile des Bildschirms die Benutzerantwort erscheint. Es ist daher eine gerechte Forderung an das aufrufende Programm, den Kursor auf den richtigen Zeilenanfang zu setzen. Sie können die Zeile für die Antwort des Benutzers löschen und den Kursor auf den Zeilenanfang setzen, indem Sie folgende Befehle verwenden:

```
2000 REM LOESCHEN DER ZEILE,AUF DER DER KURSOR STEHT
2010 PRINT CHR$(13);"□";:REM KURSOR AUF ZEILENANFANG
2020 FOR I=1 TO 39:PRINT" ";NEXT
2030 PRINT CHR$(13);"□";
2040 STOP
```

Bei einem 80spaltigen Bildschirm sollten die Befehle in Zeile 2020 statt der angegebenen 39 Leerstellen 79 schreiben.

Geben Sie dieses Programm in Ihren Computer ein; positionieren Sie den

Kursor auf eine Leerzeile zwischen zwei Textzeilen und geben Sie dann RUN <CR> zum Start des Programms ein. Wandert der gesamte Text nach oben aus dem Bildschirm heraus, dann haben Sie das Semikolon hinter dem PRINT-Befehl vergessen, der auf Zeile 2020 steht.

Häufig werden die oben angegebenen Befehle als Unterprogramm aufgerufen. In diesem Fall muß ein RETURN-Befehl auf Zeile 2040 stehen.

Alternativ zu obigem Programm können Sie auf CBM 8000-Computern die Funktionen Löschbeginn und Löschende verwenden, mit denen sich das Programm wie folgt schreibt:

```
2000 REM ZEILE LOESCHEN,AUF DER DER KURSOR STEHT
2010 PRINT CHR$(150);CHR$(22);CHR$(13);"□";
2020 STOP
```

Das Unterprogramm schrumpft zu einer einzigen Zeile, d.h. sollte nicht mehr Unterprogramm genannt werden.

Lassen Sie uns jetzt ein Unterprogramm ansehen, das Fragen stellt, die mit J für ja oder N für nein zu beantworten sind. Wir werden einen PRINT-Befehl benutzen, um eine Frage zu stellen, gefolgt von einem GET-Befehl, um einen Buchstaben als Antwort zu lesen. Löschen Sie die Zeile, auf der die Frage gestellt werden soll, mit dem Unterprogramm Zeilenlöschung. Hier das Programm mit dem verlangten Unterprogramm:

```
2000 REM ZEILE LOESCHEN,AUF DER DER KURSOR STEHT
2010 PRINT CHR$(13);"□";:REM KURSOR AUF ZEILENANFANG
2020 FOR I=1 TO 39:PRINT" ";:NEXT
2030 PRINT CHR$(13);"□";
2040 RETURN
3000 REM FRAGE STELLEN UND J/N ANTWORT IN JN$ MELDEN
3010 GOSUB 2000
3020 PRINT"WUENSCHEN SIE AENDERUNGEN?";
3030 GET JN$:IF JN$<>"N" AND JN$<>"J" THEN 3030
3040 PRINT JN$;
3050 RETURN
```

Sie können mit dem angegebenen Programm jede Frage stellen, auf die mit einer ja-nein-Antwort zu reagieren ist. Die Frage, gleich welchen Inhalts, muß im PRINT-Befehl formuliert sein, der auf Zeile 3020 steht.

Als nächstes betrachten wir einen Dialog, bei dem der Benutzer eine Zahl eingeben kann. Wir nehmen an, daß das Unterprogramm die kleinste zulässige Zahl über die Ganzzahl-Variable LO% erfährt, und die größte zulässige Zahl über HI%. Das Unterprogramm wird dann die vom Benutzer eingegebene Zahl über die Variable NM% zurückmelden. Hier das erforderliche Programm:

```
2000 REM ZEILE LOESCHEN,AUF DER DER KURSOR STEHT
2010 PRINT CHR$(13);"□";:REM KURSOR AUF ZEILENANFANG
2020 FOR I=1 TO 39:PRINT" ";:NEXT
2030 PRINT CHR$(13);"□";
2040 RETURN
3000 REM FRAGE STELLEN - ANTWORT IN FORM EINER ZAHL
3001 REM ANTWORT IN NM% ZURUECKMELDEN
3002 NM% KLEINER ALS HI% - GROESSER ALS LO%
3003 REM AUFRUFENDES PROGRAMM LIEFERT HI%,LO%
3010 GOSUB 2000
3020 PRINT"WUENSCHEN SIE AENDERUNGEN?";
3030 GET NM$:IF NM$<>"N" AND NM$<>"J" THEN 3030
3040 NM%=VAL(NM$)
3050 IF NM%<LO% OR NM%>HI% THEN 3030
3060 PRINT NM$;
3070 RETURN
```

Schreiben Sie ein kurzes Programm, das LO% und HI% Zahlenwerte zuweist und dann in Zeile 3000 des Unterprogramms springt. Fügen Sie das angegebene Unterprogramm an und starten es. Die Programmversion für CBM 8000-Computer würde den GOSUB-Befehl auf Zeile 3010 ersetzen durch den PRINT-Befehl auf Zeile 2010 des weiter oben angegebenen, 3zeiligen Programms.

Könnten Sie das Unterprogramm derart ändern, daß man auch eine zweistellige Zahl eingeben kann? Versuchen Sie, diese Programmänderung zu schreiben. Wenn es nicht geht, warten Sie bis zum nächsten Abschnitt, wo Sie dieses Programm als Teil eines Eingabeprogramms für Daten finden werden.

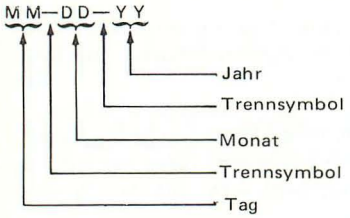
Sie können die beiden beschriebenen Dialoge noch auf eine andere einfache Weise verändern; die in Zeile 3020 ausgegebene Frage könnte bei beiden Programmen auch mittels einer Variablen für Zeichenketten vom aufrufenden Programm gestellt werden. Dadurch werden die Unterprogramme universeller verwendbar. Könnten Sie das Programm derart umschreiben, daß das Unterprogramm eine im aufrufenden Programm formulierte Nachricht aufnehmen kann?

EINGABE DES TAGESDATUMS

Die meisten Programme erfordern irgendwann relativ einfache Eingaben: mehr als ein einfaches ja—nein, aber weniger als einen Bildschirm voll Daten. Denken Sie an das Datum.

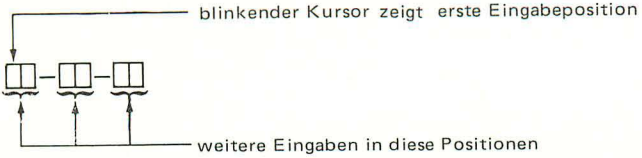
Solche einfachen Dateneingaben erfordern jedoch etwas mehr Sorgfalt, als auf den ersten Blick notwendig erscheint. Sehr wahrscheinlich wird das Datum eines unter vielen einzugebenden Daten sein. Durch sorgfältigen Entwurf der Dateneingabe für jede kleine Dateneinheit können Sie vermeiden, daß ein Benutzer die gesamte Dateneingabe zu wiederholen hat, nur weil er bei einer der Eingaben einen Fehler gemacht hatte.

Nehmen wir an, daß das Datum in folgender Form einzugeben ist:



Je nach persönlichem Stil können die zweistelligen Eingaben auch durch Schrägstrich statt Bindestrich oder ein anderes gefälliges Zeichen ersetzt werden. In der amerikanischen Datumsangabe steht der Monat am Anfang, dann kommt der Tag.

Programmieren Sie die Dateneingabe so, daß für das Auge des Benutzers ein angenehmes Bild entsteht. Der Benutzer sollte auf den ersten Blick sehen, wo Daten eingegeben werden sollen, welche Art von Daten gefragt sind und welchen Stand seine Eingabe gerade erreicht hat. Ein gutes Verfahren zur Kennzeichnung der Eingabestelle auf einem Bildschirm besteht in der Verwendung von Schriftkästchen (Datenfeldern) in Negativdarstellung. Beispielsweise gibt ein Programm zur Abfrage des Tagesdatums folgende Darstellung in Hell-Dunkel-Vertauschung aus:



Mit den folgenden Tasteneingaben <•••> sowie TAB- undCHR$- Funktionen in einem PRINT - Befehl erhalten wir eine derartige Darstellung:

```
10 PRINT " <SHIFT+ CLR> < CURSOR ↓> < CURSOR ↓>"; TAB (20) ; " <RVS> ßß
        <SHIFT  + RVS> – <RVS> ßß < SHIFT + RVS > –  <RVS> ßß
        <SHIFT + RVS> " ; CHR$ (13) ; "  <CURSOR ↑> " ; TAB (20) ;
```

ß = Symbol für Leerstellen (Leertaste) siehe auch Tabelle 4- 1 auf Seite 4- 13

Der obenstehende PRINT-Befehl kontrolliert den Kursor derart, daß die erste Datumeingabe auf Spalte 21 in Zeile 3 positioniert wird. Außerdem löscht der PRINT-Befehl den Bildschirm, so daß die Abfrage nicht von alten Textresten umgeben ist. Nach Darstellung der Eingabekästchen bewegt der PRINT-Befehl den Kursor auf das erste Eingabekästchen zurück, indem er die Befehle RETURN, CURSOR UP und TAB benutzt.

Versuchen Sie jetzt, mit einem INPUT-Befehl den eingegebenen Tag zu lesen. Zum Beispiel auf folgende Art:

```
20 INPUT TT$
```

Geben Sie Zeilen 10 und 20 als Programm ein und versuchen Sie, es zu starten. Der INPUT-Befehl wird versagen. Abgesehen von einem Fragezeichen, das die Ziffer im ersten Eingabekästchen verdrängt hat, hat der INPUT-Befehl den Rest der Zeile hinter dem Fragezeichen gelesen. Bei jeder Betätigung der RETURN-Taste erhalten Sie Nachricht RE-DO FROM START (von vorne anfangen) — es sei denn, Sie überschreiben die Eingabedarstellung durch eine sehr große Zahl.

Das ist die Gelegenheit zum Gebrauch des GET-Befehls:

```
10 PRINT"■■■";TAB(20);"■  ■■■  ■■■  ■■";CHR$(13);"■";TAB(20);
20 GET C$: IF C$="" THEN 20
30 PRINT C$; : TT$=C$
40 GET C$: IF C$="" THEN 40
50 PRINT C$; : TT$=TT$+C$
60 STOP
```

Diese Befehle lassen eine zweistellige Eingabe zu. Die Eingabe wird im ersten Feld des Datums dargestellt. Die zweistellige Eingabe erfordert keinen Wagenrücklauf (RETURN) oder ein anderes Abschlußsymbol; das Programm beendet automatisch die Dateneingabe nach der Eingabe von zwei Ziffern.

Drei zweistellige Eingaben sind erforderlich: je eine für Tag, Monat und Jahr. Statt die Programmzeilen 20 bis 50 zu wiederholen, werden wir diese Befehle zu einem Unterprogramm zusammenfassen, zu dem wir drei Mal verzweigen (Tag, Monat, Jahr):

```
10 PRINT"■■■";TAB(20);"■  ■-■  ■-■  ■";CHR$(13);"■";TAB(20);
20 GOSUB 1000 : TT$=TC$ : PRINT TAB(23)
30 GOSUB 1000 : MM$=TC$ : PRINT TAB(26)
40 GOSUB 1000 : JJ$=TC$
50 STOP
1000 REM         UNTERPROGRAMM ZUR EINGABE VON 2 SYMBOLEN
1010 GET C$: IF C$="" THEN 1010
1020 PRINT C$;
1030 GET CC$: IF CC$="" THEN 1030
1040 PRINT CC$;
1050 TC$=C$+CC$
1060 RETURN
```

Wenn Sie Besitzer eines CBM 8032-Computers sind, können Sie versuchen, das obenstehende Programm mit den Befehlen TAB SET und TAB umzuschreiben. Diese Befehle sind in Ausgabe 4.1 des Editors enthalten.

Außerdem ist die Programmversion auf einem CBM 8032-Computer viel einfacher, da man von der Funktion LÖSCHUNG BIS ZEILENENDE Gebrauch machen kann (siehe dieses Kapitel, "Eingabemöglichkeiten an CBM 8032-Computern"):

```
3000 REM FRAGE AUSGEBEN UND J/N ANTWORT IN JN$ SPEICHERN
3005 REM PROGRAMMVERSION FUER CBM 8000 COMPUTER
3010 PRINT CHR$(150);CHR$(22);CHR$(13);"■";
3020 PRINT"WUENSCHEN SIE AENDERUNGEN?";
3030 GET JN$:IF JN$<>"N" AND JN$<>"J" THEN 3030
3040 PRINT JN$;
3050 RETURN
```

Die Variablen TT$, MM$ und JJ$ enthalten die Eingaben für Tag, Monat und Jahr. Jede Eingabe wird in Form einer zweistelligen Zeichenkette gespeichert. Wie am Kapitelanfang beschrieben, sollten Sie den Tastenfeldpseicher mit seiner Speichermöglichkeit für 10 Tasteneingaben löschen, bevor die erste Eingabe erfolgt, da sonst der GET-Befehl im Unterprogramm das Zeichen einer früheren Tasteneingabe im Tastenfeldspeicher finden könnte. Sie haben den Speicher nur einmal zu leeren, und zwar vor dem ersten GET-Befehl.

Dem Benutzer kann auf zwei Arten geholfen werden, Fehler während der Eingabe des Datums zu korrigieren:

1. Das Programm kann automatisch die Richtigkeit der Eingaben für Tag, Monat und Jahr überprüfen.
2. Dem Benutzer kann die Möglichkeit zur Wiederholung der Eingabe gegeben werden.

Das Programm kann überprüfen, ob die Monatsangabe zwischen 01 und 12 liegt. Ohne sich um Schaltjahre zu kümmern, könnte das Programm auch überprüfen, ob die maximale Zahl an Tagen im Monat eingehalten wurde. Und jede Jahresangabe zwischen 00 und 99 wäre erlaubt. Jede ungültige Eingabe würde zur Wiederholung der gesamten Datumsangabe führen. Drückt der Benutzer die RETURN-Taste, dann wird die gesamte Eingabefolge wiederholt.

Unser Programm zur Datumeingabe hat schließlich folgende Form:

```
5 REM PROGRAMM ZUR EINGABE UND PRUEFUNG EINES DATUMS
10 PRINT"■■■";TAB(20);"■ ■-■ ■-■ ■";CHR$(13);"■";TAB(20);
50 GOSUB 1000:REM                          EINGABE DES TAGES
60 IF C$=CHR$(13) OR CC$=CHR$(13) THEN 10
70 DT$=TC$:PRINT TAB(23)
75 TT$=TC$
80 GOSUB 1000:REM                        EINGABE DES MONATS
90 IF C$=CHR$(13) OR CC$=CHR$(13) THEN 10
```

```
100 DT$=DT$+"-"+TC$:PRINT  TAB(26)
110 REM                PRUEFUNG AUF GUELTIGE MONATSANGABE
120 MX=VAL(TC$)
130 IF MX<1 OR MX>12 THEN 10
140 REM          WIEVIELE TAGE HAT DER ANGEGEBENE MONAT?
150 DX=31
160 IF MX=2 THEN DX=28
170 IF MX=4 OR MX=6 OR MX=9 OR MX=11 THEN DX=30
190 REM                PRUEFUNG AUF GUELTIGE TAGESANGABE
200 IF VAL(TTC$)<1 OR VAL(TTC$)>DX THEN 10
210 GOSUB 1000:REM                    EINGABE DES JAHRES
220 DT$=DT$+"-"+TC$
230 IF C$=CHR$(13) OR CC$=CHR$(13) THEN 10
240 REM                PRUEFUNG AUF GUELTIGE JAHRESANGABE
260 IF VAL(TC$)<0 OR VAL(TC$)>99 THEN 10
270 STOP
1000 REM       UNTERPROGRAMM ZUR EINGABE VON 2 SYMBOLEN
1005 FOR I=1 TO 10:GET C$:NEXT:REM TASTATURSP.LOESCHEN
1010 GET C$:IF C$="" THEN 1010
1015 IF C$=CHR$(13) THEN 1050
1016 IF C$<"0" OR C$>"9" THEN 1010
1020 PRINT C$;
1030 GET CC$:IF CC$="" THEN 1030
1035 IF CC$=CHR$(13) THEN 1050
1036 IF CC$<"0" OR CC$>"9" THEN 1010
1040 PRINT CC$;
1050 TC$=C$+CC$
1060 RETURN
```

Beachten Sie, daß aus den Eingaben Tag, Monat und Jahr eine achtstellige Zeichenkette, DT$, aufgebaut wird.

Folgende drei Prüfungen erfolgen während der Dateneingabe:

1. Ist das Eingabezeichen ein RETURN?
2. Wenn das Zeichen kein RETURN war: ist es ein gültiges Zeichen?
3. Ergeben die Zahlenkombinationen eine gültige Tagesangabe (erste Eingabe), eine gültige Monatsangabe (zweite Eingabe) und eine gültige Jahresangabe (dritte Eingabe)?

Der Wagenrücklauf (RETURN) wurde als Abbruchsymbol (Eingabewiederholung) gewählt. Indem Sie CHR$(13) auf Zeilen 60, 230, 1015, und 1035 ersetzen können Sie auch jedes andere Abbruchsymbol wählen. Betätigt der Benutzer die gewählte Taste für Abbruch, dann kann die gesamte Eingabesequenz des Datums wiederholt werden. Wir müssen im Unterprogramm für zweistellige Eingaben nach dem Abbruchsymbol abfragen (d.h. in Zeile 1035), da wir nach Eingabe der ersten oder zweiten Zahl Möglichkeit zum Abbruch haben wollen. Das Hauptprogramm (aufrufende Programm) prüft ebenfalls auf Vorliegen eines Abbruchsymbols, um dann auf Zeile 10 zurückzuspringen und die gesamte Eingabe wiederholen zu lassen. Sie könnten auch aus dem Unterprogramm für zweistellige Eingaben direkt nach Zeile 10 des aufrufenden Programms springen und dabei die Abbruchabfrage im aufrufenden Programm umgehen. Das wäre jedoch schlechte Programmierpraxis, von der wir eindringlich abraten. Jedes Unterprogramm sollte als in sich geschlossene logische Einheit behandelt werden, mit klarer Eingabestelle und Standardrücksprung ins Hauptprogramm. Innerhalb eines Unterprogramms zum Hauptprogramm zu springen, wird möglicherweise Quelle von Programmfehlern.

Wenn Sie mitten aus dem Unterprogramm in das Hauptprogramm springen, ohne über den Standardrücksprung am Ende des Unterprogramms zu gehen, setzen Sie sich allen Arten von Fehlermöglichkeiten aus, die Sie nicht verstehen, ehe Sie nicht ein ganz erfahrener Programmierer sind.

Das Unterprogramm für zweistellige Eingaben kann auch die Testlogik enthalten, mit der Eingaben von anderen Symbolen als Zahlen erkannt werden. Wir haben uns entschlossen, Symbole, die keine Zahlen darstellen, zu ignorieren. Die Befehle in Zeilen 1016 und 1036 führen eine Symbolprüfung durch, indem sie den ASCII-Wert der eingegebenen Symbole mit den ASCII-Werten der Zahlen vergleichen (siehe Anhang A über Symbolwerte im ASCII-Kode).

Eine getrennte Testlogik für Tag, Monat und Jahr müssen vorgesehen werden, da jedes dieser zweiwertigen Zeichen andere Gültigkeitsbereiche hat.

Der Befehl in Zeile 130 fragt nach der Gültigkeit der Monatsangabe ab.

Die Befehle in Zeilen 150, 160 und170 bestimmen den maximal erlaubten Tag für die Monatsangabe. Der Befehl in Zeile 200 prüft auf Gültigkeit der Tagesangabe. Der Test auf eine gültige Jahresangabe in Zeile 260 ist sehr einfach.

Beachten Sie, daß wir für die Monatsangabe eine Ganzzahl-Darstellung herstellen, nicht jedoch für Tag und Jahr. Das rührt daher, daß wir Tages- und Jahresangabe nicht sehr häufig benutzen, die Monatsangabe aber von Zeile 90 bis Zeile 170. Mit einer Ganzzahl-Darstellung des Monats verringern wir beides, den Speicheraufwand und die Programmlaufzeit.

Es kostet also mehr Zeit, ein gutes Eingabeprogramm zu schreiben, das Informationen in ansprechender Form darstellt, die Richtigkeit von Dateneingaben überprüft und dem Benutzer die Wiederholung der Eingabe jederzeit ermöglicht. Auf jeden Fall! Sie schreiben das Programm nur einmal, ein Benutzer hat es hunderte oder tausende Male zu gebrauchen. Daher investieren Sie zusätzliche Programmierzeit — einmalig, um dem Benutzer Verzögerungen bei der Programmbenutzung zu ersparen — hunderte oder tausende Male.

DATENEINGABE ÄHNLICH EINER FORMBLATTEINTRAGUNG

Besteht eine Datenmenge aus zahlreichen Einzelangaben, dann ist der beste Weg der Eingabe, den Bildschirm ähnlich wie ein Formular zu gestalten, in das die Einzelangaben einzutragen sind. Nehmen Sie eine Adressenangabe. Das Formular auf dem Bildschirm könnte folgendermaßen aussehen:

```
     EINGABEFORMULAR
1    NAME:
2 STRASSE:
3    PLZ1:              4 STADT:              5 PLZ2:
6    LAND:
```

Beachten Sie, daß jeder Eingabe eine Nummer zugeordnet wurde. Auf dem Bildschirm-Formblatt erscheinen die Zahlen in Negativschrift.

Ein Benutzer gibt die Daten aufeinanderfolgend ein, beginnend mit Eintragung 1 und endend mit Eintragung 5. Danach kann der Benutzer jede seiner Eintragungen ändern.

Das folgende Programm löscht den Bildschirm und gestaltet ihn zu einem Formblatt:

```
10 REM PROGRAMM ZUM AUFBAU EINER ADRESSKARTEI
20 REM DARSTELLUNG DES EINGABEFORMULARS
30 PRINT"■0    EINGABEFORMULAR"
40 PRINT" ■1■     NAME:"
50 PRINT" ■2■ STRASSE:"
60 PRINT" ■3■     PLZ1:";TAB(23);"■4■ STADT:";TAB(58);"■5■ PLZ2:"
70 PRINT" ■6■     LAND:"
```

Markieren Sie vor einer neuen Eingabe durch Hell-Dunkelvertauschung der Bildschirmdarstellung dasjenige Schriftfeld, in dem die Dateneingabe des Benutzers sichtbar gemacht wird. Ersetzen Sie dann das markierte Schriftfeld durch die vom Benutzer eingegebenen Zeichen. Zur Wiederholung der Dateneingabe in das laufende Schriftfeld wird die Taste CURSOR LEFT benutzt. Die Eingabe in das laufende Schriftfeld wird abgeschlossen mit der Taste RETURN. Die untenstehende Befehlsfolge enthält die notwendige Programmlogik:

```
80 REM           LIES NAMEN (MAX 25 ZEICHEN)
90 LN%=25
100 PRINT"■";TAB(11);
110 GOSUB 8000:NA$=CC$
120 REM       LIES STRASSE (MAX 25 ZEICHEN)
130 PRINT CHR$(13);TAB(11);
140 GOSUB 8000:SR$=CC$
150 REM  LIES POSTLEITZAHL1 (MAX 10 ZEICHEN)
155 LN%=10
160 PRINT CHR$(13);TAB(11);
170 GOSUB 8000:PLZ1$=CC$
180 REM          LIES STADT (MAX 25 ZEICHEN)
185 LN%=25
190 PRINT TAB(31);
200 GOSUB 8000:ST$=CC$
210 REM LIES POSTLEITZAHL2 (MAX 2 ZEICHEN)
220 LN%=2
230 PRINT TAB(65);
240 GOSUB 8000:PLZ2$=CC$
241 REM          LIES LAND (MAX 25 ZEICHEN)
242 LN%=25
243 PRINT CHR$(13);TAB(11);
244 GOSUB 8000:LND$=CC$
250 STOP
8000 REM UNTERPROGRAMM ZUM LESEN VON LN% ZEICHEN
8010 REM KURSOR MUSS AM ANFANG DES EINGABEFELDES STEHEN
8020 REM EINGABE IN EIN FELD WIRD MIT RETURN BEENDET
8030 REM WIEDERHOLUNG DER EINGABE MIT ← TASTE
8040 REM EINGABEN WERDEN NICHT AUF RICHTIGKEIT GEPRUEFT
8050 REM EINGABEN WERDEN MIT CC$ ZURUECKGEMELDET
8060 ST%=POS(X):REM KURSORSTELLUNG SPEICHERN
8070 PRINT"■";:REM EINGABEFELD IN NEGATIVDARSTELLUNG
8080 FOR I=1 TO LN%:PRINT" ";:NEXT
8090 PRINT"■";CHR$(13);"□";TAB(ST%);
8100 REM LESEN UND DARSTELLEN DER ZEICHENEINGABE
8110 CC$="":J%=0
8120 FOR I=1 TO LN%
8125 J%=J%+1
8130 GET C$:IF C$="" THEN 8130
8140 IF C$=CHR$(95) THEN PRINT CHR$(13);"□";TAB(ST%);:GOTO 8070
8150 IF C$=CHR$(13) THEN 8200
8160 PRINT C$;:CC$=CC$+C$
8170 NEXT
8190 REM ZEICHENFELD MIT LEERSTELLEN ERGAENZEN UND DARSTELLEN
8200 IF J%=LN% THEN 8300
8210 FOR I=J% TO LN%
8220 CC$=CC$+" "
8230 NEXT
8300 PRINT CHR$(13);"□";TAB(ST%);CC$;
8310 RETURN
```

Geben Sie das gesamte Programm am Tastenfeld ein, d.h. Zeile 10 bis Zeile 8310, und lassen Sie es ablaufen. Zur Erinnerung: Wenn Sie die Zahlen 10 bis 70 bereits in den Rechner eingegeben haben, brauchen Sie keine erneute Eingabe dieser Befehle.

Sollte Ihr Programm nicht richtig ablaufen, prüfen Sie sorgfältig Ihre Eingaben. Achten Sie insbesondere auf Semikolons in den PRINT-Befehlen.

Beim Programmablauf wird jedes der fünf Schriftfelder nacheinander durch Hell-Dunkel-Vertauschung der Bildschirmdarstellung markiert. Eingegebene Zeichen werden im Schriftfeld dargestellt. Wenn Sie jetzt die Taste RETURN drücken, wird die Hell-Dunkel-Vertauschung des Schriftfeldes aufgehoben und Ihre Eingabe wie gewohnt dargestellt. Versuchen Sie, die Taste CURSOR LEFT zu drücken und die Dateneingabe zu wiederholen.

Lesen Sie sorgfältig die Befehlsfolge im Unterprogramm für die Dateneingaben, beginnend mit Zeile 8060 und endend mit Zeile 8310. Bevor Sie weiter lesen, sollten Sie diese Programmlogik klar verstanden haben. Beachten Sie, wie einfach ein Benutzer seine Eingabe erkennen kann, und wie einfach jede Eingabe wiederholt werden kann, um Fehler zu korrigieren.

Nach einer abgeschlossenen Eingabe von Name und Adresse sollte das Programm den Benutzer fragen, ob sie oder er Änderungen zu machen wünschen; dann sollte das Programm fragen, welches der Schriftfelder verändert werden muß. Unterprogramme für beide Fragen wurden am Anfang dieses Kapitels angegeben. Wir werden Abwandlungen dieser Unterprogramme benutzen, in denen das aufrufende Programm Fragen an den Benutzer stellt. Löschen Sie Zeile 250 und fügen Sie die folgenden Befehle (Zeilen 260 Bis 3600) ein:

```
260 FR$="WUENSCHEN SIE AENDERUNGEN?"
270 PRINT"■■■■■■■";
280 GOSUB 3000
290 IF JN$="N" THEN STOP
300 REM  FRAGE NACH DEM ZU AENDERNDEN FELD
310 FR$="IN WELCHEM DER FELDER 1,2,3,4,5,6?"
320 LO%=1:HI%=6
330 GOSUB 3500
340 ON NM% GOTO 400,450,500,550,600,631
400 REM           AENDERUNG DES NAMENS
410 PRINT"■■■";TAB(11);:LN%=25
420 GOSUB 8000:NA$=CC$
430 GOTO 260
450 REM          AENDERUNG DER STRASSE
460 PRINT"■■■■";TAB(11);:LN%=25
470 GOSUB 8000:SR$=CC$
480 GOTO 260
500 REM   AENDERUNG DER POSTLEITZAHL1
510 PRINT"■■■■■";TAB(11);:LN%=10
520 GOSUB 8000:PLZ1$=CC$
530 GOTO 260
540 REM            AENDERUNG DER STADT
550 REM
560 PRINT"■■■■■" ;TAB(31);:LN%=25
570 GOSUB 8000:ST$=CC$
580 GOTO 260
600 REM   AENDERUNG DER POSTLEITZAHL2
610 PRINT"■■■■■" ;TAB(65);:LN%=2
620 GOSUB 8000:ZST$=CC$
630 GOTO 260
631 REM           AENDERUNG DES LANDES
632 PRINT"■■■■■■";TAB(11);:LN%=25
```

```
634 GOSUB 8000:PLZ2$=CC$
635 GOTO 260
2000 REM LOESCHEN DER ZEILE,AUF DER DER KURSOR STEHT
2010 PRINT CHR$(13);     :REM KURSOR AUF ZEILENANFANG
2020 FOR I=1 TO 39: PRINT" ";:NEXT
2030 PRINT CHR$(13);"□";
2040 RETURN
3000 REM FRAGE STELLEN ; J/N ANTWORT IN JN$ SPEICHERN
3010 GOSUB 2000
3020 PRINT FR$;
3030 GET JN$:IF JN$<>"N" AND JN$<>"J" THEN 3030
3040 PRINT JN$;
3050 RETURN
3500 REM FRAGE NACH DER FELDNUMMER 1,2,3,4,5
3510 REM ANTWORT IN NM% ZURUECKMELDEN
3520 REM NM% KLEINER ALS HI% UND GROESSER ALS LO%?
3530 REM HI%,LO%,FR$ KOMMEN VOM AUFRUFENDEN PROGRAMM
3540 GOSUB 2000
3550 PRINT FR$;
3560 GET NM$:IF NM$="" THEN 3560
3570 NM%=VAL(NM$)
3580 IF NM%<LO% OR NM%>HI% THEN 3560
3600 RETURN
8000 REM UNTERPROGRAMM ZUM LESEN VON LN% ZEICHEN
8010 REM KURSOR MUSS AM ANFANG DES EINGABEFELDES STEHEN
8020 REM EINGABE IN EIN FELD WIRD MIT RETURN BEENDET
8030 REM WIEDERHOLUNG DER EINGABE MIT ← TASTE
8040 REM EINGABEN WERDEN NICHT AUF RICHTIGKEIT GEPRUEFT
8050 REM EINGABEN WERDEN MIT CC$ ZURUECKGEMELDET
8060 ST%=POS(X):REM KURSORSTELLUNG SPEICHERN
8070 PRINT"■";:REM EINGABEFELD IN NEGATIVDARSTELLUNG
8080 FOR I=1 TO LN%:PRINT" ";:NEXT
8090 PRINT"■";CHR$(13);"□";TAB(ST%);
8100 REM LESEN UND DARSTELLEN DER ZEICHENEINGABE
8110 CC$="":J%=0
8120 FOR I=1 TO LN%
8125 J%=J%+1
8130 GET C$:IF C$="" THEN 8130
8140 IF C$=CHR$(95) THEN PRINT CHR$(13);"□";TAB(ST%);:GOTO 8070
8150 IF C$=CHR$(13) THEN 8200
8160 PRINT C$;:CC$=CC$+C$
8170 NEXT
8190 REM ZEICHENFELD MIT LEERSTELLEN ERGAENZEN UND DARSTELLEN
8200 IF J%=LN% THEN 8300
8210 FOR I=J% TO LN%
8220 CC$=CC$+" "
8230 NEXT
8300 PRINT CHR$(13);"□";TAB(ST%);CC$;
8310 RETURN
```

Lassen Sie jetzt das gesamte Programm zum Aufbau einer Adresskartei, d.h. Zeilen 10 bis 8310, mit Einfügung der Zeilen 260 bis 3600 (jedoch ohne Zeile 250) ablaufen. Hier einige Hinweise bei auftretenden Programmfehlern:

1. Wenn der Text nach oben aus dem Bildschirm hinausläuft, dann haben Sie im Unterprogramm zur Zeilenlöschung vergessen, den PRINT-Befehl mit einem Semikolon abzuschließen.
2. Wenn ein Schriftfeld in Hell-Dunkel-Vertauschung an einer falschen Stelle dargestellt wird, so haben Sie die falsche Anzahl von CURSOR DOWN-Bewegungen in einem PRINT-Befehl, oder Sie haben mit dem TAB-Befehl

zu einer falschen Spalte zugeordnet, oder Sie haben vergessen, zwei Angaben in einem PRINT-Befehl durch ein Semikolon zu trennen.

3. Wenn keine Rückmeldung auf der untersten Zeile des Bildschirms erscheint, vergewissern Sie sich, ob die im Hauptprogramm gestellten Fragen im Unterprogramm mit demselben Variablenamen aufgerufen wurden wie im Hauptprogramm.

Sie sollten das Namen- und Adressenprogramm sorgfältig studieren, um die Eingabehilfen zu verstehen, die in das Programm aufgenommen wurden. Dies sind:

1. Durch Schriftfelder in Negativdarstellung (Hell-Dunkel-Vertauschung) geben Sie dem Benutzer deutlich an, welche Eingabedaten erwartet werden und wieviele Zeichenstellen verfügbar sind.
2. Hat der Benutzer die Zahl eines zu ändernden Schriftfeldes eingegeben, dann wird ihm sofort durch Negativdarstellung des Schriftfelds zurückgemeldet, ob er das richtige bezeichnet hat.
3. Es sind nicht für alle verfügbaren Stellen eines Schriftfeldes Eintragungen erforderlich; drückt der Benutzer die RETURN-Taste, dann wird der Rest des Schriftfeldes mit Leerstellen ausgefüllt.
4. Der Benutzer kann jederzeit die Eingabe wiederholen, in dem er die CURSOR LEFT-Taste drückt.
5. Bei Fragestellungen an den Benutzer werden nur sinnvolle Zeichen als Antwort erwartet: J oder N für "JA" und "NEIN", oder eine Zahl zwischen 1 und 6 für die Auswahl eines der Schriftfelder. Es ist schlechte Programmierpraxis, andere als die einzig sinnvollen Tasteneingaben vom Programm anerkennen zu lassen. Ein Beispiel: Die Anerkennung der Eingabe J für "JA" und jedes anderen Zeichens für "NEIN" könnte verhängnisvoll sein, da die zufällige Berührung einer Taste die laufende Dateneingabe des Benutzers vorzeitig beenden würde. Umgekehrt würde die Anerkennung eines N für "NEIN" und jedes anderen Zeichens für "JA" den Benutzer unnötigerweise zwingen, erneut Daten in ein Schriftfeld einzugeben, nur weil er zufällig die falsche Taste berührt hat.

Hier einige Vorsichtsmaßnahmen bei der Dateneingabe, die wir nicht benutzt haben, die aber im Programm berücksichtigt werden könnten:

1. Überprüfen Sie die eingegebene Postleitzahl auf unzulässige Symbole.
2. Viele vorsichtige Programmierer werden zusätzlich die Frage stellen SIND SIE SICHER?, wenn ein Benutzer die Frage WÜNSCHEN SIE EINE ÄNDERUNG? mit "Nein" beantwortet. Im Falle einer falschen Tastenbedienung erhält der Benutzer hierdurch eine zweite Möglichkeit, die Frage richtig zu beantworten.
3. Wir können eine weitere Taste vorsehen, mit der die laufende Dateneingabe abgebrochen und der vorhergehende Zustand wieder hergestellt werden kann. Zum Beispiel: Wenn der Benutzer für ein zu änderndes Schriftfeld die falsche Nummer eingibt, dann muß er im Beispielprogramm das falsch bezeichnete Schriftfeld erneut eingeben. Sie könnten leicht eine weitere Taste einführen, mit der die laufende Dateneingabe abgebrochen und der ursprüngliche Zustand wieder hergestellt wird.

Versuchen Sie selbst, das Namen- und Adressprogramm zu verändern, in dem Sie die eben genannten Sicherheitsmaßnahmen hinzufügen. Als Besitzer eines CBM 8000-Computers sollten Sie außerdem versuchen, die Funktion TAB durch die Möglichkeiten von TAB SET zu ersetzen.

PROGRAMMIERUNG VON BILDSCHIRM- UND DRUCKERAUSGABEN

Nach Inbetriebnahme eines CBM-Computers erfolgt die Ausgabe auf dem Bildschirm. Nur über geeignete Befehle ist es möglich, die Ausgabe zu einem Drucker oder jedem anderen Gerät, das zum Datenempfang geeignet ist, zu senden.

Schirmbilder und Druckerbilder erfordern eine Reihe unterschiedlicher Programmiertechniken. Beispielsweise kann der Drucker breiter sein als der Bildschirm, so daß in diesem Falle die Ausgabe zwar in eine Druckerzeile paßt, aber über eine Bildschirmzeile hinausläuft. Außerdem gibt es bemerkenswerte Unterschiede in der Programmierlogik, die bei der Gestaltung eines Druckbildes anders sind als bei der Gestaltung eines Schirmbildes. Das hängt damit zusammen, daß zwar mit Kontrolltasten der Kursor über den gesamten Bildschirm bewegt werden kann, aber die gleichen Tasten nicht benutzt werden können, um den Schreibkopf eines Druckers über ein Papier zu bewegen.

Es gibt aber auch eine Reihe von Ähnlichkeiten in den Programmiertechniken zur Gestaltung von Druckbildern und Schirmbildern. Die folgende Diskussion behandelt nur die Gestaltung von Schirmbildern. Wenn Sie beabsichtigen, Programme zur Ausgabe an einen Drucker zu schreiben, sollten Sie zunächst die vorliegende Diskussion über Schirmbildausgaben lesen und dann zur Diskussion der Druckerprogrammierung in Kapitel 6 übergehen.

Die Programmierung von Bildschirmausgaben ist sehr viel einfacher als die Programmierung einer Dateneingabe, da kein Eingriff des Benutzers zu beachten ist. Sie müssen dafür sorgen, daß das Schirmbild leicht zu lesen ist, das ist alles. Hier einige Regeln zur Befolgung:

1. Vermeiden Sie die Anhäufung von zu vielen Informationen auf zu kleinem Platz.
2. Wenn Zahlen- oder Buchstabenfolgen spaltenweise aufgelistet werden, richten Sie die Daten so aus, daß der Blick leicht eine Spalte überfliegen kann.
3. Benutzen Sie die Negativdarstellung (Hell-Dunkel-Vertauschung) von Schriftfeldern, um wichtige Informationen, Überschriften und/oder Zeilenanfänge zu markieren. Verwenden Sie jedoch nicht Negativschrift für Druckerausgaben; der Drucker erzeugt eine ziemlich unlesbare Negativschrift.

Nachfolgend einige weitverbreitete Fehler, an die Sie denken sollten, wenn Sie Bildschirmausgaben programmieren:

1. Vergessen Sie nicht, die einzelnen Ausdrücke in der Liste eines PRINT-Befehls mit einem Semikolon (;) zu schließen, wenn Sie nicht eine Darstellung mit Abständen von 10 Leerzeichen wünschen, wie das Komma (,) bewirkt. Dies ist die weitverbreitetste Fehlerquelle in der Ausgabeprogrammierung.
2. Sie können viel Programmierzeit sparen, wenn Sie zuerst auf einem Blatt Millimeterpapier die Zeilen und Spalten des darzustellenden Schirmbildes markieren, ehe Sie mit der Programmierung beginnen. Das gestattet Ihnen, die Zahl der Zeilen und Spalten genau zu bestimmen. Die Alternative besteht im Ausprobieren, was letzten Endes mehr Zeit kostet.

3. Achten Sie bei der Ausgabe von Tabellen auf Situationen, in denen sich die Zahl der Tabellenwerte nicht gleichmäßig auf die vorhandene Zahl von Spalten aufteilen läßt. Zum Beispiel: Sie haben 25 Tabellenwerte in einem Datenfeld N$(I), die Sie in 3 Spalten darstellen wollen. Sie können versucht sein, das Schirmbild wie folgt zu erzeugen:

```
100 FOR I=1 TO 25 STEP 3
200 REM SCHREIBE N$(I)    IN SPALTE 1
.
.
300 REM SCHREIBE N$(I+1) IN SPALTE 2
.
.
400 REM SCHREIBE N$(I+2) IN SPALTE 3
.
.
500 NEXT I
.
600 STOP
```

Beim letzten Durchlauf der FOR-NEXT-Schleife werden die Indexwerte I = 26 und I = 27 berechnet, obwohl sie garnicht existieren. In einer FOR-NEXT-Schleife können Sie leicht das Ende einer Tabelle feststellen, indem Sie wie folgt verfahren:

```
100 FOR I=1 TO MAX STEP ST
200 REM SCHREIBE N$(I)    IN SPALTE 1
210 N=I:IF N>MAX THEN 600
.
300 REM SCHREIBE N$(I+1) IN SPALTE 2
310 N=I+1:IF N>MAX THEN 600
.
400 REM SCHREIBE N$(I+2) IN SPALTE 3
410 N=I+2:IF N>MAX THEN 600
.
500 NEXT I
.
600 STOP
```

Eine wichtige Warnung gilt Daten, die Sie von einer Diskette lesen (mit Techniken, die wir noch in Kapitel 6 beschreiben werden). CBM-Computer haben die unangenehme Angewohnheit, an das Ende von Zeichenketten, die von einer Diskette gelesen wurden, Leerstellen anzufügen. Zum Beispiel: Wenn Sie Namen auf eine Diskette schreiben und wissen, daß kein Name aus mehr als 20 Zeichen besteht, nehmen Sie an, daß beim Lesen von der Diskette jeder Name aus nach wie vor 20 Zeichen oder weniger besteht. Das ist jedoch nicht notwendigerweise der Fall. Eine unbestimmte Zahl zusätzlicher Leerstellen kann an das Ende der Buchstabenfolge angehängt worden sein. Das kann Ihr Schirmbild oder Druckerbild stören, da sich ein Schriftfeld über die nächste Spalte in Ihrer Tabellendarstellung hinaus erstrecken kann. Sie können dieses Problem vermeiden durch Benutzung der Funktion LEFT$. Ein PRINT-Befehl, der ursprünglich wie folgt lautete:

```
100 PRINT TAB(5);N$(I);TAB(30);N$(I+1)
```

nimmt nach Einfügung der Funktion LEFT$ folgende Form an:

```
100 PRINT TAB(5);LEFT$(N$(I),29);TAB(30);LEFT$(N$(I+1),20)
```

Wenn eine Liste von Variablen eine Zeichenkette von unbekannter Länge ergibt, Sie aber für alle Zeichenketten die gleiche Länge wünschen, dann müssen Sie entweder Leerstellen an das Ende zu kurzer Zeichenketten anfügen oder zu lange Zeichenketten abschneiden. Das ist einfach getan mit dem folgenden Unterprogramm:

```
10 REM VARIABLE N$ SOLL AUS 20 ZEICHEN BESTEHEN
20 REM BEI MEHR ALS 20 ZEICHEN:UEBERZAEHLIGE ZEICHEN ABTRENNEN
30 REM BEI WENIGER ALS 20 ZEICHEN:MIT LEERSTELLEN ERGAENZEN
40 LX=LEN(N$)                    : REM LX=ZAHL DER ZEICHEN IN N$
50 B$="                         ": REM RESERVOIR FUER LEERSTELLEN
60 REM IF LX>20 : ABTRENNEN UEBERZAEHLIGER ZEICHEN
70 IF LX>20 THEN N$=LEFT$(N$,20):RETURN
80 REM IF LX=20 : N$ HAT RICHTIGE LAENGE
90 IF LX=20 THEN RETURN
92 REM IF LX<20 : N$ MIT LEERSTELLEN AUS B$ ERGAENZEN
94 IF LX<20 THEN N$=N$+LEFT$(B$,20-LX):RETURN
```

Eine sehr gebräuchliche Technik bei der Behandlung großer Datenmengen ist die Darstellung eines "Fensters", in das die Daten eingetragen werden. Um ein einfaches Beispiel zu geben, werden wir ein zweidimensionales Datenfeld aus 14 Spalten und 50 Zeilen bilden. Jedes Element in diesem Datenfeld besteht aus einer vierstelligen Ganzzahl, die auf folgende Weise die Koordinaten des Elements im Datenfeld angibt:

X%(I,J) = 0I0J

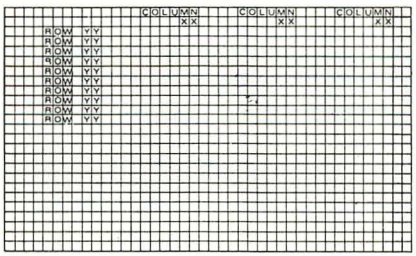
Hier einige konkrete Beispiele:

X%(3,2) = 0302
X%(19,8) = 1908
X%(11,12) = 1112
etc.

Wir können sehr einfach ein Datenfeld von 40x50-Ganzzahlen mit folgendem Programm erzeugen:

```
10 DIM XX(14,50)
20 FOR I=1 TO 14
30 FOR J=1 TO 50
40 XX(I,J)=I*100+J
50 NEXT
60 NEXT
```

Nun werden wir einen Teil dieses Datenfeldes auf dem Bildschirm darstellen. Zur Darstellung von Überschriften verwenden wir Zeilen 1 und 2 sowie Spalten 1 bis 10 des Bildschirms.



COLUMN = Spalte
ROW = Zeile

XX ist eine Zahl im Bereich 1 bis 14
YY ist eine Zahl im Bereich 1 bis 50

Hier die notwendigen Programmbefehle, mit denen die oben gezeigten Überschriften für Zeilen und Spalten in Negativschrift ausgegeben werden:

```
1000 REM UNTERPROGRAMM ZUR BESCHRIFTUNG VON ZEILEN/SPALTEN
1005 REM ALLE DARSTELLUNGEN IN NEGATIVSCHRIFT
1010 PRINT TAB(9);
1020 FOR I=1 TO 3
1030 PRINT"■    SPALTE■";
1040 NEXT
1050 PRINT CHR$(13);TAB(9);
1060 FOR I=S% TO S%+2
1070 A%=7:IF I<10 THEN A%=8
1080 PRINT  SPC(A%);"■";STR$(I);"■";
1090 NEXT
1095 PRINT  CHR$(13);
1110 FOR I=Z% TO Z%+9
1120 A%=1:IF I<10 THEN A%=2
1130 PRINT TAB(2);"■ZEILE";SPC(A%);STR$(I);"■"
1140 NEXT
1150 RETURN
```

Wir haben absichtlich ein Fenster gewählt, das kleiner als die Bildschirmgröße ist, damit wir besser die Idee eines Datenfensters illustrieren können. Es gibt jedoch keinen Grund, warum Sie nicht ein Datenfenster in Größe des gesamten Bildschirms verwenden sollten. In manchen Fällen jedoch wünscht man ein kleines Fenster, in dem nur zusammengehörige Daten auf dem Bildschirm erscheinen können.

Die Funktion STR$ zeigt eine Bildschirmdarstellung, die um eine Zeichenstelle länger ist als die Ganzzahl. Diese zusätzliche Zeichenstelle nimmt das Vorzeichen auf. Wir könnten auf das Vorzeichen verzichten. Wir haben uns entschieden, diese zusätzliche Zeichenstelle in Negativschrift auf dem Bildschirm darzustellen. Wir müssen jedoch auf das Vorzeichen Rücksicht nehmen, wenn wir die Zeichenstellen abzählen, um das Argument der Tabulatorfunktion in Zeile 1130 dieses Programms zu bestimmen.

Wir fügen jetzt Befehle hinzu, die den Benutzer zur Eingabe von 2 Zahlen für Spalte und Zeile auffordern, mit denen die Darstellung des Datenfeldes auf dem Bildschirm begonnen werden soll. Das so bezeichnete Element des Datenfeldes wird links oben im Datenfenster dargestellt. Danach wird das Datenfenster mit den Elementen aus benachbarten Spalten und Zeilen aufgefüllt. Erweitern Sie Ihr Programm um folgende Zeilen:

```
5 REM PROGRAMM ZUR DARSTELLUNG EINES TABELLEN-AUSSCHNITTS
6 REM GROESSE DES DARSTELLUNGSFENSTERS VOM BENUTZER BESTIMMT
10 DIM X%(14,50)
20 FOR I=1 TO 14
30 FOR J=1 TO 50
40 X%(I,J)=I*100+J
50 NEXT
60 NEXT
64 PRINT"■";
65 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■■";
70 INPUT "BITTE SPALTE EINGEBEN (1 BIS 12):";S%
80 IF S%<1 OR S%>12 THEN PRINT "■";:GOTO 70
90 INPUT " BITTE ZEILE EINGEBEN (1 BIS 41):";Z%
100 IF Z%<1 OR Z%>41 THEN PRINT"■";:GOTO 90
105 PRINT "■";:GOSUB 1000
110 PRINT"■■■";
120 FOR I=Z% TO Z%+9
130 PRINT TAB(9)
```

```
140 FOR J=SX TO SX+2
150 X$=STR$(X%(J,I))
155 PRINT SPC(10 -LEN(X$));X$;
160 NEXT
165 PRINT CHR$(13);
170 NEXT
180 PRINT"■■■■■■■FORTSETZUNG? J ODER N ?"
190 GET C$: IF C$<>"J" AND C$<>"N" THEN 190
200 IF C$="J" THEN 65
210 STOP
```

Lassen Sie das Programm ablaufen. Wenn Sie das Programm richtig eingegeben haben, werden Sie als erstes feststellen, daß der Computer stoppt und scheinbar für einige Zeit nichts tut; er ist beschäftigt mit der Ausführung der verschachtelten FOR-NEXT-Befehle, die auf den Zeilen 20 bis 60 stehen. Es kostet ungefähr 10 bis 15 Sekunden, bis das Datenfeld X% mit Ganzzahlen gefüllt ist.

Der PRINT-Befehl in Zeile 64 löscht zunächst den Bildschirm, so daß alle Textreste entfernt sind, ehe die INPUT-Befehle in den Zeilen 70 und 90 den Benutzer nach Zeile und Spalte fragen, mit denen die Darstellung begonnen werden soll. Wir nehmen diesen Befehl zur Bildschirmlöschung nicht in den PRINT-Befehl in Zeile 65 auf, da das Programm zur Zeile 65 zurückkehrt, um nach neuen Zeilen- und Spaltenzahlen zu fragen, vor deren Eingabe wir nicht die vorangehende Bildschirmdarstellung löschen wollen.

Beachten Sie, daß bei der Eingabe die Spaltenzahlen 1 bis 12 zulässig sind; 3 Spalten werden auf dem Bildschirm dargestellt, so daß bei Eingabe der Spaltenzahl 12 gerade die verfügbare Spaltenzahl 14 des Datenfeldes ausgeschöpft wird. Ähnlich ergibt sich die zulässige Eingabe der Zeilenzahlen 1—41, da von den 50 verfügbaren Zeilen des Datenfeldes 10 auf dem Bildschirm dargestellt werden, so daß die höchsten Zeilenzahlen 41—50 sind.

In Zeile 150 des Programms wird jedes Element des Datenfeldes vom Typ Ganzzahl, X%, umgewandelt in den Typ Zeichenkette, X$, bevor es in Zeile 155 ausgegeben wird. Der Zeilenabstand kann jetzt einfach aus der Länge der Zeichenkette berechnet werden, wie die Funktion LEN im PRINT-Befehl in Zeile 155 zeigt.

Beachten Sie, wie der Abfrage des Benutzers durch die Befehle in Zeilen 70, 90 und 180 Programmstufen folgen, die jede ungültige Eingabe verhindern. Sogar in einem so einfachen Programmbeispiel nehmen wir uns die Zeit, eine sichere Eingabe zu programmieren.

Das Programm zur Darstellung des Ausschnitts eines Datenfeldes im Bildschirmfenster erfährt eine nützliche Erweiterung, wenn dem Benutzer die Möglichkeit gegeben wird, das Bildschirmfenster nach oben, unten, links oder rechts zu bewegen. Das ist schnell getan. Von den verfügbaren Standardsymbolen auf einem CBM-Tastenfeld werden wir das Pik-Symbol benutzen, um das Bildschirmfenster um eine Zeile nach oben zu bewegen, wodurch sich die erste dargestellte Zeilennummer um 1 verringert. Für die Abwärtsbewegung des Bildschirmfensters um 1 Zeile werden wir das Herz-Symbol benutzen, wodurch sich die erste dargestellte Zeilenzahl um 1 erhöht. Für die Linksbewegung werden wir das Symbol KLEINER ALS benutzen, wodurch sich das Bildschirmfenster im Datenfeld um eine Spalte nach links bewegt (die erste dargestellte Spaltenzahl wird um 1 verringert). Für die Rechtsbewegung benutzen wir das Symbol GRÖSSER ALS, wodurch der Ausschnitt des Datenfeldes sich um eine Spalte nach rechts verschiebt und sich die erste dargestellte Spaltenzahl um 1 erhöht. Um dies alles zu erreichen, müssen wir die Befehle in Zeilen 180 bis 210 durch die nachfolgenden Befehle ersetzen:

```
180 PRINT"■■■■■■■FORTSETZUNG?  ↑,↓,<,>  ODER  J,N  EINGEBEN
190 GET C$: IF C$="" THEN 190
200 REM    FUER C$="↑" : TABELLENZEILE UM 1 VERRINGERN
210 IF C$="↑" THEN Z%=Z%-1:PRINT CHR$(13);"■";:GOTO 100
220 REM    FUER C$="↓" : TABELLENZEILE UM 1 ERHOEHEN
230 IF C$="↓" THEN Z%=Z%+1:PRINT CHR$(13);"■";:GOTO 100
240 REM    FUER C$=<    : TABELLENSPALTE UM 1 VERINGERN
250 IF C$="<" THEN S%=S%-1:GOTO 320
260 REM    FUER C$=>    : TABELLENSPALTE UM 1 ERHOEHEN
270 IF C$=">" THEN S%=S%+1:GOTO 320
280 REM    FUER C$=J : NEUE EINGABE, FUER C$=N : STOP
290 IF C$="J" THEN 65
300 IF C$="N" THEN STOP
310 GOTO 190:REM ZURUECKWEISUNG UNERLAUBTER ANTWORTEN
320 IF S%<1 OR S%>12 THEN PRINT CHR$(13);:GOTO 105
330 GOTO 105
```

Beachten Sie die Einfachheit der Programmlogik, obwohl wir noch immer die Eingaben auf Benutzerfehler überprüfen. Jede andere Eingabe außer den sechs erlaubten Zeichen wird zurückgewiesen. Wenn die Änderung der Zeilen- oder Spaltenzahl das Bildschirmfenster über den Rand des Datenfeldes hinausbewegt, dann fragt die Programmlogik einfach nach der Eingabe neuer Zeilen- und Spaltenzahlen. (CBM 8000-Computer erlauben die Programmierung von Bildschirmfenstern, jedoch sind diese Möglichkeiten im vorliegenden Beispiel nicht besonders nützlich, da wir das Bildschirmfenster nicht nur nach oben und unten, sondern auch nach links und rechts verschieben wollen.)

Das oben wiedergegebene Programm enthält eine kleine Inkonsequenz: Beim Überschreiten der zulässigen Zeilenzahl kann lediglich die Eingabe einer neuen Zeilenzahl erfolgen; das hängt mit den Befehlen GOTO 100 in den Zeilen 210 und 230 zusammen. Überschreitet die Spaltenzahl den zulässigen Bereich, dann erlaubt der Befehl GOTO 70 in Zeile 300 sowohl die Eingabe einer neuen Spalte als auch einer neuen Zeile (da der Abfrage der Spaltenzahl jedesmal die Abfrage der Zeilenzahl folgt, nicht jedoch umgekehrt, wie Zeilen 70 und 90 im Hauptteil des Programms zeigen). Könnten Sie das Programm derart umschreiben, daß entweder nur die Zeile oder nur die Seite neu eingegeben werden können? Oder beide, Zeile und Spalte, neu eingegeben werden können, wenn eine von ihnen den zulässigen Bereich überschritten hat?

Eine andere unerwünschte Eigenschaft des Programms ist die Zeit, die es benötigt, um das Datenfeld X% anzulegen. Das hat nichts zu tun mit dem Bildschirm. Derartige Verzögerungen erfahren Sie in vielen Programmen. Ein Benutzer könnte annehmen, daß sein Computer nicht richtig funktioniert, und wenn derartige Wartezeiten auftreten, ist es ein guter Gedanke, mit einer auffälligen Nachricht dem Benutzer mitzuteilen, daß sein Computer arbeitet und er bitte noch warten soll. Das ist schnell getan. Sie lassen einfach den Befehlen zur Berechnung des Datenfeldes einen geeigneten PRINT-Befehl vorangehen. Folgender PRINT-Befehl könnte in unserem Falle benutzt werden:

```
15 PRINT"■ BITTE WARTEN: DIE TABELLE WIRD ANGELEGT"
```

Unser Programm achtet sehr darauf, die Bildschirmdarstellung mit der 39. Spalte zu beenden und nicht auch noch die 40. und letzte Spalte zu beschreiben. Bei CBM-Computern mit 40spaltigen Bildschirmen ist es nicht ratsam, sämtliche 40 Spalten auszunutzen. Sie können Gefahr laufen, die Zeilensprunglogik anzusprechen, die bei mehr als 40 Zeilen die Darstellung auf der folgenden Zeile fortsetzt. Sie halten am besten Ihre eigenen, programmierten Zeilensprünge (RETURN) fern von Zeilensprüngen, die automatisch am Zeilenende erfolgen.

ZEILENSPRUNG BEI 40-SPALTIGEN SCHIRMEN

Der folgende Abschnitt erklärt die Zeilensprunglogik bei 40spaltigen Bildschirmen.

Jede Programmzeile kann auf CBM-Computern bis zu 80 Zeichen enthalten, die sich im Falle eines 40spaltigen Bildschirms auf zwei Bildschirmzeilen verteilen. Solange jedoch weniger als 40 Zeichen in einer Bildschirmzeile stehen, nimmt der Computer an, daß die Programmzeile aus 39 Zeichen besteht. Bei Eingabe des 40. Zeichens in eine Bildschirmzeile (wobei der Kursor auf den Anfang der folgenden Bildschirmzeile springt) nimmt der Computer an, daß die Programmzeile aus 79 Zeichen bestehen soll und er sich augenblicklich in der ersten Hälfte dieser Programmzeile befindet.

Trifft ein Programm auf einen Wagenrücklauf (RETURN-Befehl), dann führt es einen Zeilensprung zur nächsten logischen Bildschirmzeile aus. Im Falle der kurzen Programmzeile (weniger als 40 Zeichen je Bildschirmzeile) erfolgt der Zeilensprung auf die nachfolgende Bildschirmzeile. Befindet sich dagegen der Computer in der ersten Hälfte einer langen Programmzeile (d.h. die augenblickliche Bildschirmzeile enthält 40 Zeichen), dann erfolgt der Zeilensprung auf die nächste logische Bildschirmzeile, die sich zwei Zeilen tiefer befindet.

Stellen Sie jedoch bei einem 40spaltigen Bildschirm das 40. Zeichen mit einem POKE-Befehl dar, dann geht der Computer nicht von der Annahme aus, daß eine Programmzeile mit 79 Zeichen vorliegt. Ein derartiger POKE-Befehl hat folgende Form:

```
POKE 32767+(L-1)*40,CH$
```

wobei:

L Zeilennummer
CH$ darzustellendes Zeichen

Besitzt Ihr Computer einen 40spaltigen Bildschirm, dann ist es ein Versuch wert, das vorher behandelte Tabellenprogramm derart zu ändern, daß es bis zur 40. Spalte einer Bildschirmzeile reicht. Hierfür ist folgendes zu tun: das Argument der TAB-Befehle in Zeilen 30 und 1010 ist von 9 auf 10 zu erhöhen; das Argument des TAB-Befehls in Zeile 1050 muß von 13 in 14 geändert werden; das Argument des TAB-Befehls in Zeile 1130 verändert sich von 2 in 3. Versuchen Sie jetzt einen Programmlauf; die Zahlen des Datenfeldes werden spaltenweise dargestellt, aber es treten zuviele Wagenrückläufe auf mit der Folge, daß Teile der Darstellung nach oben aus dem Bildschirm gedrängt werden. Der Versuch, die zusätzlichen Wagenrückläufe zu unterdrücken und ein richtiges Schirmbild zu erzeugen, stellt eine schwierige Programmieraufgabe dar.

RECHNEN MIT MEHR ALS 9-STELLIGEN ZAHLEN

CBM-Computer können die Operationen Addition, Subtraktion, Multiplikation und Division mit bis zu 9stelligen Zahlen ausführen. Die Begrenzung auf 9 Stellen kann jedoch Probleme schaffen. Beispielsweise können Ganzzahlen die Werte 0 – 999999999 annehmen; in Dollar und Cent ergibt das den Bereich $ 0,00 bis $ 9,999,999.99 (oder $ 999,999,999, wenn Cent nicht benötigt wird). Dezimalzahlen können im Bereich von 0,000000001 bis zu 99999999.0 dargestellt werden. Obwohl

diese Grenzen in vielen Anwendungen keine Schwierigkeiten darstellen, wird es häufig Zahlen geben, die diese Grenzen sowohl im kommerziellen wie im wissenschaftlichen Bereich überschreiten.

Zwei Programmiermethoden können die Beschränkung der numerischen Länge von CBM-Computern überwinden. Die erste Methode benutzt numerische Zeichenketten. Die zweite Methode benutzt eine Parallelarithmetik, bei der eine große Zahl in kleinere Blöcke zerlegt und jeder Block getrennt der arithmetischen Operation unterzogen wird.

ADDITION

Numerische Zeichenketten und Parallelarithmetik können beide benutzt werden, um Ganzzahlen zu addieren, die aus mehr als 9 Stellen bestehen. Von den beiden Summanden wird die erste Zahl Augend und die zweite Zahl Addend genannt. Der Addend wird zum Augend addiert.

ADDITION MIT NUMERISCHEN ZEICHENKETTEN

Diese Addition erfolgt über folgende Schritte:

1. Eingabe von Augend und Addend als zwei positive numerische Zeichenketten.
2. Rechtsbündige Verschiebung der Zeichenketten.
3. Getrennte, stellenrichtige Addition der Zahlen beider Zeichenketten unter Beachtung des Überlaufs.
4. Verkettung der Antworten zu einer einzigen Zeichenkette, die das Ergebnis darstellt.
5. Darstellung der resultierenden Zeichenkette.

Lassen Sie uns nacheinander jeden Schritt untersuchen.

Schritt 1: Eingabe von Augend und Addend als positive numerische Zeichenkette mit Hilfe eines INPUT-Befehls:

Bildschirm-Darstellung

```
10 PRINT"■***ADDITION***":PRINT
20 INPUT A$,B$

RUN

***ADDITION***

?1234567890123456
??57943572
```

Speicherinhalte

A$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

B$ | 5 | 7 | 9 | 4 | 3 | 5 | 7 | 2 |

Die Variable A$ ist der Augend, die Variable B$ ist der Addend. Der INPUT-Befehl erlaubt beiden Summanden, die Zahlenlänge von 9 Stellen zu überschreiten. Der Einfachheit halber werden wir lediglich positive Ganzzahlen als Eingabe zulassen. Sobald Sie vertraut mit der Grundidee des Additionsprogramms sind, sollten Sie versuchen, das Programm derart zu ändern, daß es auch die Addition mit negativen Zahlen und Bruchzahlen ausführen kann.

Schritt 2: Rechtsbündige Verschiebung der Zeichenketten.

Vor der Ausführung arithmetischer Operationen sollten die Zahlen zuerst rechtsbündig verschoben werden, da in BASIC alphabetische und numerische Zeichen automatisch linksbündig verschoben sind. Wurden die Zahlenwerte der numerischen Zeichen-

ketten addiert, ohne daß erst eine rechtsbündige Verschiebung stattgefunden hat, dann sind die Antworten falsch, wie das folgende Beispiel zeigt:

Falsch: linksbündig verschoben	**Richtig:** rechtsbündig verschoben
1234567890123456	1234567890123456
+57943572	+ 57943572
7028925090123456	1234565948067028

Die folgenden Befehle bewirken eine Rechtsverschiebung der kürzeren der beiden numerischen Zeichenketten. Die kürzere Zeichenkette wird mit führenden Nullen ergänzt, bis beide Zeichenketten gleiche Länge haben. X stellt die Länge der Zeichenkette A$ dar. Y stellt die Länge der Zeichenkette B$ dar:

```
30 BLANK$ = "                ": REM 16 LEERSTELLEN
40 X=LEN(A$):Y=LEN(B$)
50 IF X<Y THEN A$=LEFT$(BLANK$,Y-X)+A$
60 IF Y<X THEN B$=LEFT$(BLANK$,X-Y)+B$
```

BLANK$ in Zeile 30 dient als Reservoir für Leerstellen, mit denen die kürzere Zeichenkette aufgefüllt wird. Da wir uns auf eine Zahlenlänge von 16 Stellen beschränkt haben, enthält BLANK$ 16 Leerstellen.

In Zeile 40 wird die Länge X, Y der Zeichenketten bestimmt. In Zeilen 50 und 60 werden die Zeichenketten X, Y verglichen und die kürzere der beiden Zeichenketten mit führenden Nullen aufgefüllt. Hierfür wird die Funktion LEFT$ verwendet, die von der Zeichenkette BLANK$ soviele Leerstellen liest, wie der Längendifferenz Y – X (bzw. X – Y) entspricht. Leerstellen werden mit dem folgenden Befehl vor die kürzere der beiden Zeichenketten gesetzt (im Beispiel wird B$ als die kürzere Zeichenkette angenommen):

```
LEFT$(BLANK$(X-Y)+B$
```

Dieser Befehl bewirkt folgende Veränderungen:

B$=LEFT$(BLANK$,X–Y)	+B$
B$=LEFT$(BLANK$,16–8)	+B$
B$=LEFT$(BLANK$,8)	+B$
B$=LEFT$(ØØØØØØØØØØØØØØØØ,8)	+B$
B$=ØØØØØØØØ	+ 57943572
B$=ØØØØØØØØ57943572	

A$=1234567890123456	B$=ØØØØØØØØ57943572
16 Stellen	16 Stellen

Schritt 3: Stellenrichtiges Addieren der Zeichenketten.

Auf den ersten Blick könnten Sie annehmen, daß die Zeichenketten A$ und B$ mit dem folgenden Befehl addiert werden können:

```
C$=A$+B$
```

Das ist jedoch falsch. Wird ein Additionszeichen zwischen Zeichenketten verwendet, dann werden sie nicht addiert, sondern verkettet:

```
C$=A$+B$
```

C$=1234567890123456 + ØØØØØØØØ57943572

C$=1234567890123456ØØØØØØØØ57943572

Wir wollen jedoch die Stellen der Zeichenketten addieren und nicht verketten. Um numerische Zeichenketten zu addieren, müssen wir jede einzelne Zahl einer

Zeichenkette auslesen, sie in eine numerische Konstante umwandeln und zu der entsprechenden Zahl der anderen Zeichenkette addieren. Dies wird mit den beiden Zeichenkettenfunktionen VAL und MID$ ausgeführt:

```
1020 FOR I=LEN(A$) TO 1 STEP-1
1030 A=VAL(MID$(A$,I,1))
1050 B=VAL(MID$(B$,I,1))
1100 NEXT I
```

A in Zeile 1030 liest nacheinander die Ziffern in der Zeichenkette A$ aus. Entsprechendes gilt für B und B$. Der Anfangswert des Zeigers I ist gleich der Länge der Zeichenketten (sowohl A$ als auch B$ können als Argument in der Funktion LEN benutzt werden). Bei jedem Durchlaufen der Schleife FOR-NEXT wird der Zähler I um 1 verringert, so daß die Funktionen MID$ nacheinander und von rechts nach links die Ziffern einer Zeichenkette auslesen können:

```
1        MID$(B$,I,1)

16  ØØØØØØØØ57943572
15  ØØØØØØØØ57943572
14  ØØØØØØØØ57943572
13  ØØØØØØØØ57943572
12  ØØØØØØØØ57943572
11  ØØØØØØØØ57943572
10  ØØØØØØØØ57943572
 9  ØØØØØØØØ57943572
 8  ØØØØØØØØ57943572
 7  ØØØØØØØØ57943572
 6  ØØØØØØØØ57943572
 5  ØØØØØØØØ57943572
 4  ØØØØØØØØ57943572
 3  ØØØØØØØØ57943572
 2  ØØØØØØØØ57943572
 1  ØØØØØØØØ57943572
```

Mit der Funktion VAL wird jede ausgelesene Ziffer als numerischer Wert dargestellt:

Für I = 16,

B=VAL(MID$(B$,16,1))

B=VAL($ØØØØØØØ57943572)

B=Ø2

Für I = 15,

B=VAL(MID$(B$,15,1)) . . .

Nachdem die Ziffern beider Zeichenketten in Ganzzahlen umgewandelt worden sind, werden sie addiert und ihre Summen als Variable C$ ausgegeben. Hier die erforderlichen Programmschritte:

```
1000 N=1

1010 D=0

1020 FOR I=LEN(A$) TO 1 STEP -1

1030 A=VAL(MID$(A$,I,1))

1040 A=A+D:D=0

1050 B=VAL(MID$(B$,I,1))
```

```
1060 C=A+B

1070 IF C>=10 THEN D=1

1080 IF D=1 AND I=1 THEN N=2

1090 C$=RIGHT$(STR$(C),N)+C$

1100 NEXT I
```

Die Variable D in Zeile 1010 wird zunächst auf den Wert 0 gesetzt; danach nimmt D in den Zeilen 1040, 1070 und 1080 den Überlauf auf. Ein Übertrag tritt auf wenn die Summe C in Zeile 1060 größer als oder gleich 10 ist. Die Zehner-Stelle wird als Übertrag in D gespeichert und bei der Addition der nachfolgenden Ziffer berücksichtigt:

```
1070 IF C>=10 THEN D=1
```

A $\boxed{\text{ø}6}$
+B $\boxed{\text{ø}9}$
C $\boxed{\text{ø}15}$ ⟶ 15 >= 10 → D $\boxed{\text{ø}1}$

or

A $\boxed{\text{ø}3}$
+B $\boxed{\text{ø}0}$
C $\boxed{\text{ø}3}$ ⟶ 3 < 10 → D $\boxed{\text{ø}0}$ (unverändert)

Da die Summe zweier einstelliger Zahlen höchstens den Wert 18 erreicht, wobei der Übertrag 1 ist, kann D entweder nur 0 oder 1 sein, aber niemals größer als 1.

Der Überlauf in D wird beim nächsten Schleifendurchlauf berücksichtigt:

```
1040 A=A+D:D=0
```

Würde man diesen Befehl vergessen, dann würde der Überlauf nie berücksichtigt werden, und A wäre falsch. In Vorbereitung der nächsten Schleifenwiederholung wird D nach der Addition von A und D auf den Wert 0 zurückgesetzt.

Schritt 4: Verkettung der Einzelsummen C und Umwandlung der Gesamtsumme in eine Zeichenkette. Die Summanden Augend und Addend wurden als Zeichenketten eingegeben, um die Beschränkung auf 9 Stellen zu vermeiden. Aus dem gleichen Grund müssen auch die Einzelsummen in Zeichenketten verwandelt werden.

In Zeile 1090 werden die Einzelsummen C verkettet und als Zeichenkette ausgegeben.

Die Funktion STR$ (C) verwandelt C in eine Zeichenkette. Diese Zeichenkette besteht aus zwei Zeichen: Einem Vorzeichen und einem Zahlenwert. Dieses Vorzeichen stört jedoch bei der "Addition" von Zeichenketten. Mit der Funktion RIGHT$ wird daher nur der Zahlenwert (der Betrag) der Zeichenkette STR$ extrahiert:

```
1000 N=1
```

N $\boxed{\text{ø}1}$

```
1060 C=A+B
```

C $\boxed{\text{ø}8}$ = A $\boxed{\text{ø}6}$ + B $\boxed{\text{ø}2}$

```
1090 C$=RIGHT$(STR$(C),N)+C$
```

C$=RIGHT$(STR$(C),1)+C$

C$=RIGHT$($\boxed{\text{ø}8}$,1)+C$

C$= $\boxed{8}$ + C$

Selbst wenn C eine zweistellige Zahl ist, wird nur die rechte Ziffer mit C$ verkettet. Dies wird bestimmt vom Index N in der Funktion RIGHT$. Dieser Index wird in Zeile 1000 auf den Wert 1 gesetzt. Lediglich im letzten Schleifendurchlauf wird N auf den Wert 2 gesetzt, falls ein Überlauf vorlag, d.h. D = 1. Ohne Zeile 1080 würde ein Überlauf D nicht mehr im letzten Schleifendurchlauf berücksichtigt werden, da seine Addition in Zeile 1040 nicht mehr erfolgt. Daher werden im letzten Schleifendurchlauf beide Stellen der Einzelsumme C beachtet.

```
1070 IF C>=10 THEN D=1
        C[ø12]>=10     D[ø1]
1080 IF D=1 AND I=1 THEN N=2
        D[ø1]          I[ø1]          N[ø2]
1090 C$=RIGHT$(STR$(C),N)+C$
     C$=RIGHT$([ø12],2)+C$
     C$=[12]+C$
     C$=[12XXXXXXX]
```

Die vollständige FOR-NEXT-Schleife in Zeilen 1020 bis 1100 arbeitet wie folgt:

1. Sie extrahiert einzelne Ziffern von einer numerischen Zeichenkette und ordnet ihnen numerische Werte zu (Befehle in Zeilen 1030, 1050).
2. Die Ziffern zweier Zeichenketten werden nacheinander addiert (Zeile 1060) und auf einen Übertrag abgefragt (Zeile 1070). Dieser Übertrag wird bei der nachfolgenden Addition berücksichtigt (Zeile 1040).
3. Die Einzelsummen werden dann verkettet und als numerische Zeichenkette dargestellt (Zeile 1090).

Schritt 5: Die Darstellung der Ergebnis-Summe.
Zum Abschluß dieses Additionsprogramms werden der FOR-NEXT-Schleife Befehle vorangesetzt, mit denen die Art und Länge der Eingaben geprüft werden (Zeilen 10 bis 1010). Hinzu kommen die Befehle PRINT und CLEAR (Zeilen 1110 bis 1130). Das Programm hat schließlich folgende Formen:

```
15 PRINT"□*** A D D I T I O N ***":PRINT
20 INPUT A$,B$
30 BLANK$="                    "
40 X=LEN(A$):Y=LEN(B$)
50 IF X<Y THEN A$=LEFT$(BLANK$,Y-X+A$)
60 IF Y<X THEN B$=LEFT$(BLANK$,X-Y)+B$
1000 N=1
1010 D=0
1020 FOR I=LEN(A$) TO 1 STEP -1
1030 A=VAL(MID$(A$,I,1))
1040 A=A+D:D=0
1050 B=VAL(MID$(B$,I,1))
1060 C=A+B
1070 IF C>=10 THEN D=1
1080 IF D=1 AND I=1 THEN N=2
1090 C$=RIGHT$(STR$(C),N)+C$
1100 NEXT I
1110 PRINT:PRINT"ERGEBNIS= ";C$
1120 C$="":PRINT:GOTO 20
1130 END
```

Zwei Versuchsläufe des Programms ergaben folgende Ergebnisse:

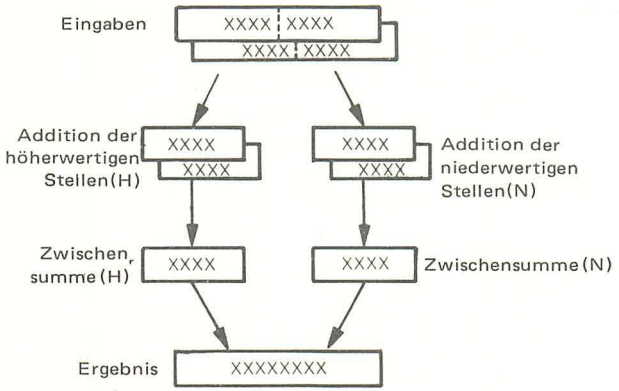
```
*** A D D I T I O N ***

? 12345
?? 579

ERGEBNIS= 12924

? 1234567890123456
?? 57943572

ERGEBNIS= 1234567948067028
```

Dieses Additionsprogramm überwindet die Beschränkung auf neunstellige Zahlen. Versuchen Sie, das Programm derart zu verändern, daß Sie Dollar und Cent eingeben können und das Ergebnis im gleichen Format erhalten.

ADDITION IN PARALLELARITHMETIK

Die Beschränkung auf neunstellige Zahlen der Addition kann durch Verwendung der Parallelarithmetik umgangen werden.

Parallelarithmetik teilt eine große Zahl in kleinere Zahlenblöcke auf. Jeder Block wird unabhängig bearbeitet. Die Einzelergebnisse werden dann zu einem Schlußergebnis vereinigt, wie folgendes Schema zeigt:



Addition in Parallelarithmetik erfolgt mit folgenden Schritten:

1. Eingabe von Augend und Addend als zwei positive numerische Zeichenketten.
2. Teilung der Zahl in zwei gleich große Zahlhälften.
3. Ausführung der Summationen getrennt für die höherwertigen und niederwertigen Zahlhälften.
4. Verkettung der Teilsummen zur Ergebnissumme.
5. Darstellung des Ergebnisses als Zeichenkette.

Schritt 1: Eingabe von Augend und Addend als zwei positive numerische Zeichenketten:

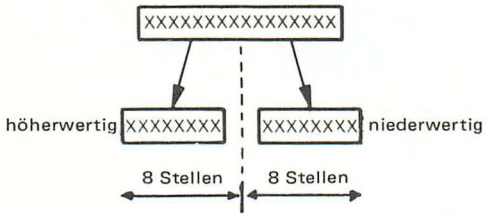
```
15 PRINT"■*** ADDITION IN PARALLELARITHMETIK ***":PRINT
20 INPUT A$,B$
```

```
RUN

*** ADDITION IN PARALLELARITHMETIK ***

? 1234567890123456
?? 57943572
```

A$ ist der Augend und B$ der Addend. Die Zahlen werden als numerische Zeichenketten eingegeben, da es für Zeichenketten Funktionen gibt, mit denen eine Zahlenreihe in kürzere Zahlenblöcke geteilt werden kann, und damit die Beschränkungen der Zahlenlänge im Computer umgangen werden.

Schritt 2: Bestimmen der größten eingegebenen Zahlenlänge.
Aus der größten Zahlenlänge ergibt sich die Anzahl der Zahlenblöcke, in die die eingegebenen Zahlen geteilt werden müssen. Ein Beispiel: Bei einer Zahlenlänge von 16 Stellen werden die Zahlen in zwei Zahlenblöcke mit einer größten Länge von 8 Stellen je Zahlenblock geteilt.

Um unser Programmbeispiel einfach zu halten, wurde als größte Zahlenlänge 16 Stellen angenommen. Die eingegebene Zahl wird in einen höherwertigen und einen niederwertigen Zahlenblock geteilt:



Zunächst müssen wir bestimmen, welche der Zahlen länger ist, indem wir die Länge ihrer entsprechenden Zeichenketten A$ und B$ bestimmen. Die Länge von A$ ist X, die Länge von B$ ist Y:

```
1000 X=LEN(A$):Y=LEN(B$)
```

Die größere der beiden Zahlenlängen X, Y wird halbiert und ergibt die Länge F eines Zahlenblockes. Hierzu sind folgende Vergleiche erforderlich:

```
1002 IF X>Y THEN F=X/2:GOTO 1006
1004 F=Y/2
```

Hier eine einfachere Anweisung zur Bestimmung des Wertes von F:

```
1002 F=Y/2:IF X>Y THEN F=X/2
```

Für die Zahlenbeispiele A$ = "1234567890123456" und B$ = "57943572" erhalten wir folgendes:

```
1000 X=LEN(A$):Y=LEN(B$)
     X = 16         Y = 8
1002 IF X>Y THEN F=X/2:GOTO 1006
```

16>8, daher F=16/2 = 8
das Programm fährt in Zeile 1006 fort

Sobald der Wert von F gefunden ist, fährt das Programm in Zeile 1006 fort. Der Befehl in Zeile 1006 stellt fest, ob F eine Bruchzahl ist. F wird auf den nächst höheren ganzzahligen Wert aufgerundet. Hierzu wird zunächst der ganzzahlige Anteil in F bestimmt und dann der Wert 1 addiert. Ein Beispiel: Der Wert F = 7,1 wird in Zeile 1006 in F = 8 aufgerundet:

```
1006 IF F>INT(F) THEN F=INT(F)+1
```

Beispiel: wenn 7,5 > 7 dann F = 7+1 = 8

Die Zeichenketten A$ und B$ werden nun in Felder der Länge F zerlegt. A$ zerfällt in die Felder AH, AN, während B$ in die Felder BH, BN zerfällt. H bezieht sich auf die höherwertigen Stellen, N auf die niederwertigen Stellen der geteilten Zahl. Die Felder werden mit folgenden Befehlen gebildet:

```
1000 X=LEN(A$):Y=LEN(B$)
1002 IF X>Y THEN F=X/2:GOTO 1006
1004 F=Y/2
1006 IF F>INT(F) THEN F=INT(F)+1
1010 IF X<=F THEN AH=0:AN=VAL(A$):GOTO 1040
1020 AH=VAL(LEFT(A$,X-F))
1030 AN=VAL(RIGHT$(A$,F))
1040 IF Y<=F THEN BH=0:BN=VAL(B$):GOTO 1070
1050 BH=VAL(LEFT$(B$,Y-F))
1060 BN=VAL(RIGHT$(B$,F))
```

In Zeilen 1010 und 1040 stehen Befehle, die die Länge X, Y der Zeichenketten mit der Feldlänge F vergleichen (in unserem Beispiel ist F = 8). Ist die Zeichenkette kürzer als F, dann werden AH bzw. BH auf den Wert 0 gesetzt, womit nur AN bzw. BN zur Darstellung der Zeichenketten A$ bzw. B$ übrigbleiben. Ist die Länge der Zeichenketten jedoch größer als die Feldlänge F, dann erfolgt eine Teilung in zwei Felder. Die Felder AN, BN sind gleich den F-Stellen der rechten Seiten von A$, B$, wie die Befehle in Zeilen 1030 und 1060 zeigen. Die Felder AH, BH sind gleich dem verbleibenden linksseitigen Rest von A$, B$, wie Zeilen 1020 und 1050 angeben. Für unser Zahlenbeispiel sieht die Bestimmung von AH wie folgt aus:

```
1020 AH=VAL(LEFT$(A$,X-F))
```

AH=VAL(LEFT$(A$,16−8))

AH=VAL(LEFT$(1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 ,8))

AH=VAL(1 2 3 4 5 6 7 8)

AH=ø12345678

Im gleichen Beispiel wird AN wie folgt aus A$ bestimmt:

```
1030 AN=VAL(RIGHT$(A$,F))
```

AN =VAL(RIGHT$(1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 ,8))

AN =VAL(9 0 1 2 3 4 5 6)

AN =ø90123456

Das gleiche Vorgehen gilt für BH und BN. Beachten Sie, daß die Funktion VAL Zeichenketten in Zahlen verwandelt.

Schritt 3: Addition der Felder AH, BH und AN, BN.
Sobald lange Zahlenreihen in Felder geteilt wurden, die kurz genug sind, um vom CBM-Computer bearbeitet zu werden, kann die Addition beginnen. Die Felder können wie ganz normale Zahlen addiert werden, da sie von der Funktion VAL von einer Zeichenkette in eine Zahlendarstellung umgewandelt worden sind. Es ist also keine stellenweise Addition erforderlich wie bei numerischen Zeichenketten. Die Teilsummen CH und CN lauten in unserem Zahlenbeispiel wie folgt:

AH	ø12345678	AN	ø90123456
+BH	ø00000000	+BN	ø57943572
CH	ø12345678	CN	ø148067028

Erst werden die Felder AN und BN mit folgendem Befehl addiert:

```
1070 CN$=STR$(AN+BN)
```

Sie Summe von AN und BN wird wieder in eine numerische Zeichenkette verwandelt und durch CN$ dargestellt. Diese Umwandlung ist nicht unbedingt erforderlich, aber das Vorhandensein eines Übertrags in der Summe CN$ ist einfacher mit der für Zeichenketten gültigen Funktion LEN zu prüfen.

In Zeile 1080 der unten wiedergegebenen Programmfortsetzung wird die Länge der Zeichenkette CN$ mit der Feldlänge F verglichen. Ist die Länge von CN$ größer als F, dann ist die hinzugekommene Stelle durch einen Übertrag entstanden, der auf die Summe CH$ addiert werden muß. Ein Übertrag wird in D durch den Wert 1 angezeigt.

Die Teilsumme CH$ entsteht aus der Addition von AH, BH und dem Übertrag D.

Vor der Verkettung der Teilsummen CH$ und CN$ werden führende Leerstellen abgeschnitten. Erinnern Sie sich, daß diese Leerstellen ein nicht dargestelltes Pluszeichen bedeuten und bei der Umwandlung von Zahlen in Zeichenketten erhalten bleiben. Bei der Verkettung von CH$ und CN$ wünschen wir jedoch nicht den Einschluß von Leerstellen. In Zeilen 1075 bzw. 1095 werden die Leerstellen mit Hilfe der Funktionen MID$ abgeschnitten:

```
1070 CN$=STR$(AN+BN)
     CN$=STR$(ø90123456 + ø57943572)
     CN$=STR$(ø148067028)
     CN$=ø148067028
1075 CN$=MID$(CN$,2,LEN(CN$)-1)
     CN$=MID$(ø148067028,2,10-1)
     CN$=MID$(ø148067028,2,9)
     CN$=148067028
1080 IF LEN(CN$)) > F THEN D=1
     LENCN$=9   F=8
     9>8→D=1
1090 CH$=STR$(AH+BH+D)
     CH$=STR$(ø12345678 + ø00000000 + ø1)
     CH$=STR$(ø12345679)
     CH$=ø12345679
1095 CH$=MID$(CH$,2,LEN(CH$)-1)
     CH$=MID$(ø12345679,2,10-1)
     CH$=MID$(ø12345679,2,9)
     CH$=12345679
```

Schritt 4: Verkettung der Teilsummen CH$, CN$.
Eine in CN$ mögliche Übertragsstelle wurde bereits berücksichtigt, so daß wir bei der Verkettung mit CH$ nur noch an den rechten acht Stellen in CN$ interessiert sind:

```
1100 C$=CH$+RIGHT$(CN$,F)
C$=CH$ø12345679 + RIGHT$ (CN$ ø148067028,8)
C$=ø12345679 + 48067028
C$=ø1234567948067028
```

Schritt 5: Ausgaberesultat C$.

```
1110 PRINT:PRINT"ANSWER=";C$:PRINT
```

Das Programm ist jetzt vollständig. Die Addition in Parallelarithmetik erlaubt die Eingabe von zwei positiven Ganzzahlen mit bis zu 16 Stellen. Die Zahlen werden dann in zwei kürzere Felder von je 8 Stellen geteilt. Die Felder werden getrennt summiert und die Teilsummen zum Resultat C$ verkettet, das bis zu 17 Stellen enthalten kann. Dieses Additionsprogramm erlaubt Ihnen das Arbeiten mit Zahlen, die um 8 Stellen länger sind als die maximal zulässige Zahlenlänge in CBM-Computern.

Nachfolgend eine Auflistung des gesamten Programms und das Beispiel eines Programmlaufs:

```
10 PRINT"*** ADDITION IN PARALLELARITHMETIK ***":PRINT
20 INPUT A$,B$
1000 X=LEN(A$):Y=LEN(B$)
1002 IF X>Y THEN F=X/2:GOTO 1006
1004 F=Y/2
1006 IF F>INT(F) THEN F=INT(F)+1
1010 IF X<=F THEN AH=0:AN=VAL(A$):GOTO 1040
1020 AH=VAL(LEFT$(A$,X-F))
1030 AN=VAL(RIGHT$(A$,F))
1040 IF Y<=F THEN BH=0:BN=VAL(B$):GOTO 1070
1050 BH=VAL(LEFT$(B$,Y-F))
1060 BN=VAL(RIGHT$(B$,F))
1070 CN$=STR$(AN+BN)
1075 CN$=MID$(CN$,2,LEN(CN$)-1)
1080 IF LEN(CN$)>F THEN D=1
1090 CH$=STR$(AH+BH)
1095 CH$=MID$(CH$,2,LEN(CH$)-1)
1100 C$=CH$+CN$
1110 PRINT"ERGEBNIS= ";C$:PRINT
1120 AH=0:AN=0:BH=0:BN=0:D=0:CH$="":CN$="":C$="":GOTO 20
1130 END
```

```
*** ADDITION IN PARALLELARITHMETIK ***


?1234567890123456
??57943572


ERGEBNIS= 1234567948067028
```

Versuchen Sie eine Abwandlung des Programms, mit der Zahlen als Dollar und Cent eingegeben und Summen dargestellt werden können.

SUBTRAKTION

Wie bei der Addition kann man Zahlen mit mehr als 9 Stellen subtrahieren, indem man numerische Zeichenketten oder die Technik der Parallelarithmetik verwendet.

SUBTRAKTION MIT NUMERISCHEN ZEICHENKETTEN

Das Subtraktionsprogramm enthält viele Abschnitte des Programms "Addition mit numerischen Zeichenketten". Die einzelnen Schritte sind wie folgt:

1. Eingabe von Minuend und Subtrahend als zwei positive numerische Zeichenketten.
2. Rechtsbündige Verschiebung der Zeichenketten.
3. Bestimmung der längeren der beiden Zeichenketten.
4. Stellenweise Subtraktion der Zeichenketten unter Beachtung des Übertrags.
5. Verkettung der Teilantworten zu einer Ergebnis-Zeichenkette.
6 Unterdrückung führender Nullen in der Ergebnis-Zeichenkette.
7. Ausgabe des Ergebnisses.

Schritt 1: Eingabe von Minuend und Subtrahend.
Beide Zahlen werden von einem INPUT-Befehl in Form positiver numerischer Zeichenketten angegeben:

```
10 PRINT"***SUBTRACTION***":PRINT
20 INPUT A$,B$

RUN

***SUBTRACTION***

?123456789012                    A$ 1 2 3 4 5 6 7 8 9 0 1 2
??57943572                       B$ 5 7 9 4 3 5 7 2
```

A$ ist der Minuend, d.h. die erste der eingegebenen Zahlen, von der die andere Zahl abzuziehen ist. B$ ist der Subtrahend, d.h. die Zahl, die vom Minuenden zu subtrahieren ist.

Schritt 2: Rechtsbündige Verschiebung von Minuend und Subtrahend.
Dieser Vorgang ist der gleiche wie in Schritt 2 des Programms "Addition mit numerischen Zeichenketten":

```
30 BLANK$="                                "
40 X=LEN(A$):Y=LEN(B$)
50 IF X<Y THEN A$=LEFT$(BLANK$,Y-X)+A$
60 IF Y<X THEN B$=LEFT$(BLANK$,X-Y)+B$
```

Schritt 3: Vergleich der Zahlenwerte von Minuend und Subtrahend.
Für die Subtraktion müssen wir bestimmen, welche der beiden Zahlen die größere ist. Obwohl die eingegebenen Zeichenketten die gleiche Länge haben können, kann ihr Zahlenwert sehr verschieden sein.

Die Zahlenwerte A$ und B$ werden mit Hilfe der Funktion VAL in den Zeilen 65 und 70 verglichen:

```
65 IF VAL(A$)=VAL(B$) THEN C$="0":GOTO 1150
70 IF VAL(A$)>VAL(B$) GOTO 1000
```

Danach wird der Wert von B$ von dem Wert von A$ subtrahiert.
Wenn der Wert von A$ größer als der Wert von B$ ist, haben wir es mit einer einfachen Subtraktion zu tun, und das Programm springt auf Zeile 1000.

Wenn der Wert von B$ größer ist als der Wert von A$, dann haben wir eine größere Zahl von einer kleineren abzuziehen, und das Programm muß auf die Ausgabe eines negativen Ergebnisses vorbereitet sein. Das Programm vertauscht zunächst den Inhalt von A$ und B$, so daß der Zahlenwert in A$ wieder größer ist als in B$. Das Ergebnis der Subtraktion, C$, erhält ein negatives Vorzeichen, das der Zeichenkette C$ vorangesetzt wird: C$ = "–"+C$".

Ein Beispiel: A$ habe den Zahlenwert 3, B$ den Zahlenwert 5, d.h. VAL (B$)>VAL (A$), oder der Subtrahend ist größer als der Minuend:

A$ [3]
B$ [5]

A$ und B$ vertauschen:

A$ [3] B$ [5] → A$ [5] B$ [3]

Subtrak tion: VAL(A$)–VAL(B$)=C$
A$ [5] – B$ [3] → C$ [2]

Negative Vorzeichen anfügen:

C$ = "–" + C$
"–" + C$ [2] → C$ [–2]

Antwort:

C$ [–2]

Die Vertauschung der Zahlenwerte von A$, B$ erfolgt in Zeile 80:

```
80 X$=A$:A$=B$:B$=X$
```

Programmanweisung	Speicherinhalt		
	X$	A$	B$
⋮	0	3	5
X$=A$	3	3	5
A$=B$	3	5	5
B$=X$	3	5	3

X$ dient als Zwischenspeicher. Ohne X$ würde zwar der ursprüngliche Inhalt von A$ mit B$ überschrieben werden, zum Überschreiben von B$ wäre dann aber der Wert von A$ verloren gegangen:

Programmanweisung	Speicherinhalt		
	A$	B$	
⋮	3	5	
A$=B$	5	5	falsch
B$=A$	5	5	

Im weiteren Verlauf des Programms müssen wir wissen, ob die Eingangsvariablen ausgetauscht wurden. Wir setzen deshalb eine Flagge, die angibt, daß die Zahlen von A$ und B$ ausgetauscht wurden. Diese Flagge heißt S: S behält den Wert 0, wenn die Zahlen nicht ausgetauscht wurden; S wird auf den Wert 1 gesetzt, wenn der Austausch stattgefunden hat. Die Flagge S wird im Programm Zeile 90 gesetzt:

```
90 S=1
```

Wurde die Flagge auf den Wert 1 gesetzt, dann muß mit Hilfe der Verkettungsoperationen für Zeichenketten ein Minuszeichen vor das Ergebnis C$ "addiert" werden, ehe es ausgegeben wird. Diese Verkettung erfolgt in Zeile 1140:

```
1140 IF S=1 THEN C$="-"+C$
```

Schritt 4: Ausführung der einfachen Subtraktion.

Nach dem vorhergehenden Schritt ist der Wert von A$ immer größer als der Wert von B$, unabhängig vom Vorzeichen. Wir können jetzt eine einfache Subtraktion durchführen, die in Zeilen 1000 und 1080 angegeben ist. Die Operationen wurden direkt von den Zeilen 1020 bis 1100 im Schritt 3 des Programms "Addition mit numerischen Zeichenketten" übernommen, da dort das Auslesen von Zahlen aus einer Zeichenkette in gleicher Weise erfolgt. In Zeile 1050 des genannten Programms wird jetzt jedoch die Überlaufvariable D für den negativen Überlauf (oder die Entlehnung) benutzt. Für (A – B)<0 muß eine Zehnerstelle von der linken Nachbarstelle entlehnt oder geborgt werden, wodurch sich A um den Wert 10 vergrößert. Die Überlaufvariable D wird auf den Wert –1 gesetzt, da eine "1" entlehnt wurde, wodurch sich der Zahlenwert der linken Nachbarspalte verringert hat. Das Ergebnis wird C genannt:

```
1000 REM**SUBTRACTION ROUTINE**
1010 FOR I=LEN(A$) TO 1 STEP-1
1020 A=VAL(MID$(A$,I,1))
1030 A=A+D:D=0
1040 B=VAL(MID$(B$,I,1))
1050 IF (A-B)<0 THEN D=-1:A=A+10
1060 C=A-B
```

Dieses Programm führt eine stellenweise, rechtsbeginnende Subtraktion durch, die am folgenden Zahlenschema sichtbar gemacht werden soll:

										+9		
				+10	+10	+10				+10	+10	
A	1	2	3	3	4	5	7	8	8	0	1	2
–B	0	0	0	0	5	7	9	4	3	5	7	2
C	1	2	3	3	9	8	8	4	5	4	4	0

Schritt 5: Bildung des Ergebnisses.

In unserem Subtraktionsprogramm werden die Ergebnisse der stellenweise Subtraktion laufend mit den vorangehenden Ergebnissen verkettet, bis das Gesamtergebnis resultiert. Diese fortlaufende Verkettung kann direkt von Zeile 1090 des Programms "Addition mit numerischen Zeichenketten" genommen werden; da die Stellenzahl des Subtraktionsergebnisses sich nie durch einen Übertrag erhöht, können wir auf die Variable N verzichten. In Zeile 1070 unseres Programms ist die fortlaufende Verkettung wie folgt enthalten:

```
1070 C$=RIGHT$(STR$(C),1)+C$
```

Schritt 6: Unterdrückung führender Nullen.

Die Subtraktion kann führende Nullen in der Antwort verursachen. Vor der Ausgabe des Ergebnisses unterdrücken wir diese führenden Nullen. Die Programmschleife FOR-NEXT in Zeile 1090 bis 1120 unterdrückt alle führenden Nullen mit der Funktion VAL:

```
1090 FOR I=1 TO LEN(C$)
1100 IF VAL(MID$(C$,I,1))=0 THEN L=L+1
1110 IF VAL(LEFT$(C$,I))<>0 THEN I=LEN(C$)
1120 NEXT I
```

Die FOR-NEXT-Schleife durchsucht das Subtraktionsergebnis C$ der Länge nach nach führenden Nullen oder Leerstellen, indem es jede Ziffer in C$ ausliest und mit der Ziffer 0 vergleicht. Dieser Ziffernvergleich läuft von links nach rechts. Wird eine 0 oder Leerstelle erkannt, dann erhöht das Programm den Zähler L um 1 (Zeile 1100). Sobald die erste Ziffer gefunden wurde, die nicht 0 ist, wird der Schleifenzähler L auf den Wert der Gesamtlänge von C$ gesetzt, so daß das Programm sofort die Programmschleife verlassen kann.

Sobald die Zahl der führenden Nullen und Leerstellen bestimmt worden ist, sondern wir sie vom Rest der Antwort C$ ab. Dies erfolgt in Zeile 1130, in der die Funktion RIGHT$ die ursprüngliche Länge von C$, LEN(C$), um die überflüssigen Nullen, L, kürzt und damit das Subtraktionsergebnis C$ bildet:

C$ = 0 0 1 2 3 5 7 LEN(C$) = 7

I	MID$(C$,I,1)		
1	0 0 1 2 3 5 7	= 0	L = 1
2	0 0 1 2 3 5 7	= 0	L = 2
3	0 0 1 2 3 5 7	< > 0	I = LEN(C$)
7			I = 7 Verlassen der Schleife

```
1130 C$=RIGHT$(C$,LEN(C$)-L)
```

C$=RIGHT$ (0 0 1 2 3 5 7,7-2)

C$=RIGHT$ (0 0 1 2 3 5 7,5)

C$= 1 2 3 5 7

Schritt 7: Ausgabe des Subtraktionsergebnisses C$.
Vor Ausgabe von C$ haben wir noch zu prüfen, ob das Ergebnis negativ ist, indem wir die Variable S in Programm-Zeile 1140 abfragen. Ist S = 1, d.h. A$ < B$, dann hat das Ergebnis negativ zu sein, und wir müssen ein Minuszeichen zu C$ hinzufügen. Ist S = 0, dann ist das Ergebnis positiv und nichts muß hinzugefügt werden. Mit Zeile 1150 erfolgt schließlich die Ergebnisdarstellung:

```
1140 IF S=1 THEN C$="-"+C$
1150 PRINT"ERGEBNIS= ";C$:PRINT
```

Die letzten Zeilen 1160 bis 1180 setzen alle vom Programm benutzten Zeichenketten und Variablen auf den Wert 0 und bringen das Programm für eine neue Zahleneingabe an den Anfang zurück. Hier die Auflistung des vollständigen Programms:

```
10 PRINT"□*** S U B T R A K T I O N ***":PRINT   Bildschirm löschen
20 INPUT A$,B$                                   Eingabe
30 BLANK$="                    "
40 X=LEN(A$):Y=LEN(B$)
50 IF X<Y THEN A$=LEFT$(BLANK$,Y-X)+A$           rechtsbündige Verschiebung
60 IF Y<X THEN B$=LEFT$(BLANK$,X-Y)+B$           der Zeichenketten
65 IF VAL(A$)=VAL(B$) THEN C$="0":GOTO 1150
70 IF VAL(A$)>=VAL(B$) GOTO 1000.                für A$ < B$ Vertauschen
80 X$=A$:A$=B$:B$=X$                             beider Zeichenketten
90 S=1
1000 REM SUBTRAKTIONS-ROUTINE
1010 FOR I=LEN(A$) TO 1 STEP -1                  Subtraktions-Schleife
1020 A=VAL(MID$(A$,I,1))
1030 A=A+D:D=0
1040 B=VAL(MID$(B$,I,1))
```

```
1050 IF (A-B)<0 THEN D=-1:A=A+10
1060 C=A-B
1070 C$=RIGHT$(STR$(C),1)+C$
1080 NEXT I
1090 FOR I=1 TO LEN(C$)
1100 IF VAL(MID$(C$,I,1))=0 THEN L=L+1
1110 IF VAL(LEFT$(C$,I))<>0 THEN I=LEN(C$)
1120 NEXT I
1130 C$=RIGHT$(C$,LEN(C$)-L)
1140 IF S=1 THEN C$="-"+C$
1150 PRINT:PRINT"ERGEBNIS= ";C$:PRINT
1160 C$="":A$="":B$="":X$=""
1165 A=0:B=0:C=0:S=0:X=0:Y=0
1170 GOTO 20
1180 END
```

Subtraktions-Schleife

Unterdrücken führender Nullen und Leerstellen

Antwort ausgeben

Rücksetzen für neuen Programmlauf

```
*** S U B T R A K T I O N ***

? 123456789012
?? 57943572

ERGEBNIS= 123398845440
```

Das wiedergegebene Programm bereitet jedoch noch eine Schwierigkeit: Es liefert als Ergebnis 0, wenn Subtrahend und Minuend die gleiche Stellenzahl haben und überdies in den ersten 9 Stellen identisch sind. Ein Beispiel: Versuchen Sie die Subtraktion mit den Zahlen 123456789000 und 123456789012. Es wird das falsche Ergebnis 0 ausgegeben. Dieser Fehler rührt von den Befehlen in Zeile 65 her. Die Funktion VAL rechnet einen neunstelligen Wert für die numerischen Zeichenketten A$ und B$. Stimmen diese Zeichenketten in den neun ersten Stellen überein, dann stellt der Test in Zeile 65 Gleichwertigkeit von A$ und B$ fest, auch wenn sie in den restlichen Stellen nicht übereinstimmen. Können Sie dieses Problem beseitigen, indem Sie für die obere und untere Hälfte der numerischen Zeichenketten getrennte Tests einführen?

SUBTRAKTION DURCH PARALLELARITHMETIK

Erinnern Sie sich von der zurückliegenden Diskussion, daß es das Verfahren der Parallelarithmetik ist, eine lange Zahl in kürzere Zahlenfelder zu teilen, die Rechnung für jedes Zahlenfeld getrennt auszuführen und anschließend die Teilantworten zum Ergebnis zu verketten? Dieses Verfahren umgeht die Beschränkung auf neunstellige Zahlen.

Eine Subtraktion in Parallelarithmetik besteht in folgenden Schritten:

1. Eingabe von Minuend und Subtrahend als zwei positive numerische Zeichenketten.
2. Erkennen der Zeichenkette mit dem höheren Zahlenwert.
3. Teilung der Zahlung in höherwertige und niederwertige Hälften.
4. Getrennte Subtraktion der höherwertigen und niederwertigen Hälften.
5. Verkettung der Teilergebnisse zum Endergebnis.
6. Unterdrückung führender Nullen.
7. Ergebnisdarstellung.

Schritt 1: Eingabe von Minuend und Subtrahend.
Beide Zahlen werden über den INPUT-Befehl in Zeile 20 als positive numerische Zeichenketten eingelesen:

```
10 PRINT"■*** SUBTRAKTION IN PARALLELARITHMETIK ***":PRINT
20 INPUT A$,B$
```

```
RUN

*** SUBTRAKTION IN PARALLELARITHMETIK ***

?123456789012
?? 57943572
```

Der Minuend A$ und Subtrahend B$ werden als numerische Zeichenketten eingegeben, um in CBM-Computern die Begrenzung der Zahlendarstellung auf neun Stellen zu umgehen.

Wie bei der Addition in Parallelarithmetik werden A$ und B$ in kürzere Felder geteilt. Die Länge der eingegebenen Zahlen wurde willkürlich auf maximal 16 Stellen begrenzt, so daß wir die längste numerische Zeichenkette in gleiche Zahlfelder von je 8 Stellen teilen können.

Schritt 2: Bestimmung der Zeichenkette mit dem größeren Zahlenwert. Sind die Zahlenwerte von A$ und B$ gleich, dann springt das Programm in Zeile 1190 und gibt als Ergebnis eine 0 aus. Ist B$ größer als A$, dann ist das Ergebnis negativ und besondere Schritte sind erforderlich.

Ist ein negatives Ergebnis zu erwarten, dann werden die Zahlen in A$ und B$ vertauscht, so daß der größere Zahlenwert in A$ und der kleinere Zahlenwert in B$ steht. Danach erfolgt die Subtraktion und die Verkettung des Ergebnisses C$ mit einem vorangestellten Minuszeichen, was wir schon in Zeile 70 der "Subtraktion mit numerischen Zeichenketten" demonstriert haben. Zeile 30 dient dazu, die Zahlenvertauschung von A$, B$ zu überspringen, falls sie nicht erforderlich ist:

```
30 IF VAL(A$)>VAL(B$) THEN 1000
40 X$=A$:A$=B$:B$=X$
50 S=1
```

Ist der Zahlenwert B$ größer als der Zahlenwert von A$, dann erfolgt in Zeile 40 und 50 eine Zahlenvertauschung zwischen A$ und B$. Hierdurch wird sichergestellt, daß die kleinere Zahl von der größeren subtrahiert wird. Bei erfolgter Zahlenvertauschung wird die Flagge S auf den Wert 1 gesetzt.

Eine ausführliche Beschreibung dieser Befehlsfolge finden Sie unter Schritt 3 im Programm "Subtraktion mit numerischen Zeichenketten".

Schritt 3: Teilung von A$ und B$ in höherwertige und niederwertige Hälften.

```
1000 X=LEN(A$):Y=LEN(B$)
1002 IF X>Y THEN F=X/2:GOTO 1006
1004 F=Y/2
1006 IF F>INT(F) THEN F=INT(F)+1
1010 IF X<=F THEN AH=0:AN=VAL(A$):GOTO 1040
1020 AH=VAL(LEFT$(AH,X-F))
1030 AN=VAL(RIGHT$(A$,F))
1040 IF Y<=F THEN BH=0:BN=VAL(B$):GOTO 1070
1050 BH=VAL(LEFT$(B$,Y-F))
1060 BN=VAL(RIGHT$(B$,F))
```

Die Länge F der kürzeren Zahlfelder wird in Zeilen 1002 und 1006 bestimmt. Diese Zeilen sind identisch mit den Zeilen 1002 und 1006 in dem Programm "Addition in Parallelarithmetik". In Zeilen 1010 und 1040 wird die Länge der Zeichenketten A$, B$ mit der Feldlänge F verglichen. Ist die Zeichenkette kürzer als F, dann wird das höherwertige Feld AH (oder BH) auf den Wert 0 gesetzt, so daß nur die niederwertigen Felder AN (oder BN) zur Darstellung der Zeichenkette A$ (oder B$) übrigbleiben. Ist eine Zeichenkette jedoch länger als F, dann muß sie in ein höherwertiges und ein niederwertiges Zahlenfeld geteilt werden. Als Beispiel: AH wird aus den linksseitigen Stellen in A$ gebildet, die nach Abzug der Feldlänge F übrigbleiben:

A$ 1|2|3|4|5|6|7|8|9|0|1|2

B$ 5|7|9|4|3|5|7|2

AH ø123456 AN ø789012

BH øøøøø57 BN ø943572

Die Teilung von A$ in die Felder AH, AN und entsprechend B$ in BH, BN benutzt die schon im Programm "Addition in Parallelarithmetik" beschriebenen Programmzeilen 1000 bis 1060.

Schritt 4: Getrennte Subtraktion der höherwertigen und niederwertigen Felder.
Die Subtraktionen erfolgen einerseits zwischen AH und BH und andererseits zwischen AN und BN:

AH ø123456 AN ø789012

−BH øøøøø57 BN ø943572

Vor der Subtraktion der Felder müssen Minuend und Subtrahend verglichen werden. Ist der Zahlenwert von BN größer als der Zahlenwert von AN, ergibt sich eine negative Differenz CN. Ein negatives CN kann jedoch nicht ohne Probleme mit CH verkettet werden:

CH ø xxxxxx ⌒ CN − xxxxxx = C ø xxxxxx − xxxxxx falsch

Um eine negative Differenz CN zu vermeiden, müssen wir den Zahlenwert von AN erhöhen, indem wir eine Stelle von AH borgen. Das Borgen einer Stelle, um damit AN zu erhöhen, erfolgt in Zeilen 1070 bis 1090. Die Zeilen 1080 und 1090 können übersprungen werden, wenn AN größer ist als BN. In diesem Falle springt das Programm direkt zur Subtraktion in Zeile 1100:

```
1070 IF AN>=BN THEN 1100
1080 AN=AN+10↑F
1090 AH=AH-1
```

Ist BN größer als AN, dann müssen wir in unserem Zahlenbeispiel den Wert 1 Millionen von AH borgen, um damit den Wert von AN zu erhöhen:

AH xxxxxx	AN 1 xxxxxx
−BH xxxxxx	−BN xxxxxx
CH xxxxxx	CN xxxxxx

Der negative Übertrag (geborgte Stelle) läßt sich ganz allgemein als Zehnerpotenz mit der Feldlänge F als Exponenten darstellen:

AN=AN + 10↑F

In unserem Programmbeispiel ist AN kleiner als BN, wie in Zeile 1070 festgestellt:

AN ø789012 BN ø943572

Bei einer Feldlänge F = 6 müssen wir uns daher den Wert 10↑F = 10↑6 = 1.000.000 von AH borgen, um damit den Wert von AN zu erhöhen:

```
1080 AN=AN+10↑F
```

AN=AN+1016

AN=AN+1000000

AN = 789012 + 1000000

AN = 1789012

Nach Erhöhung von AN müssen wir AH um den negativen Übertrag −1 verringern, da wir diesen Wert von AH geborgt haben.

```
1090 AH=AH-1
     AH=ø123456 - ø1
     AH=ø123455
```

Nach diesen Vorbereitungen werden die Felder AH, AN, BH und BN subtrahiert. CN$ ist die Differenz von AN und BN, und CH$ ist die Differenz von AH und BH.

CN$ wird in den Programmzeilen 1100 bis 1102 berechnet:

```
1100 CN$=STR$(INT(AN-BN))
CN$=STR$(ø1789012-ø943572)
CN$=STR$(ø845340)
CN$=ø345540
```

In Zeile 1101 wird mittels der Funktion MID$ die links von CN$ stehende Leerstelle abgetrennt (diese Stelle steht für ein Pluszeichen):

```
1101 CN$=MID$(CN$,2,LEN(CN$)-1)
CN$=MID$(ø845440,2,6)
CN$=  845440
```

In Zeile 1102 wird CN$ mit führenden Nullen aufgefüllt, falls es kürzer als die Feldlänge F ist. Zur Verkettung führender Nullen mit CN$ wird zunächst in Zeile 15 eine Nullen-Zeichenkette ZERO$ angelegt. In unserem Beispiel ist die Länge von CN$ = F, so daß führende Nullen nicht erforderlich sind:

```
15 ZERO$="0000000000000000000"
1102 CN$=LEFT$(ZERO$,F-LEN(CN$))+CN$
CN$=LEFT$ (ZERO$,6-6)+ CN$
CN$=LEFT$ (ZERO$,0)+ CN$
```

CH$ wird in Zeile 1110 berechnet:

```
1110 CH$=STR$(INT(AH-BH))
     CH$=STR$(ø123455-ø57)
     CH$=STR$(ø123398)
     CH$=ø123395
```

Mittels der Funktion MID$ wird eine linksstehende Leerstelle abgetrennt:

```
1111 CH$=MID$(CH$,2,LEN(CH$)-1)
     CH$=MID$(ø123398,2,6)
     CH$= 123398
```

Die Operationen des Subtraktionsprogramms lassen sich an unserem Zahlenbeispiel wie folgt demonstrieren:

```
1070 IF AN>=BN GOTO 1100
     789012 >=943572 ————————→ falsche Anweisung

1080 AN=AN+10↑F
     AN=789012+1000000
     AN=1789012
```

```
1090 AH=AH-1
     AH=123456-1
     AH=123455
1100 CN$=STR$(INT(AN-BN))
     CN$ =STR$(ø1789012-ø943572)
     CN$ =STR$(ø845540)
     CN$ =8455440
1101 CN$=MID$(CN$,2,LEN(CN$)-1)
     CN$ =MID$(ø8455440,2,7-1)
     CN$ =MID$(ø8455440,2,6)
     CN$=8455440
1102 CN$=LEFT$(ZERO$,F-LEN(CN$))+CN$
     CN$=LEFT$(ZERO$,6-6)+CL$
     CN$=LEFT$(ZERO$,0)+8455440
     CN$=8455440
1110 CH$=STR$(INT(AH-BH))
     CH$=STR$(ø123455-ø57)
     CH$=STR$(ø123398)
     CH$=ø123398
1111 CH$=MID$(CH$,2,LEN(CH$)-1)
     CH$=MID$(ø123398,2,7-1)
     CH$=MID$(ø123398,2,6)
     CH$=123398
```

Schritt 5: Verkettung der Teilergebnisse CH$ und CN$.

In Zeile 1120 erfolgt die Verkettung der numerischen Zeichenketten CH$ und CN$:

```
1120 C$=CH$+CN$
C$=CH$[    ]+CN$[    ]
C$= [          ]
```

Schritt 6: Unterdrückung führender Nullen in C$.

Vor der Ausgabe C$ an den Bildschirm werden führende Nullen unterdrückt. Für die Erkennung führender Nullen wurde die Programmschleife FOR-NEXT aus Schritt 6 der zuvor besprochenen "Subtraktion mit numerischen Zeichenketten" übernommen. Außerdem wurde aus Schritt 7 die Verkettung C$ mit einem Minuszeichen für den Fall eines negativen Resultats übernommen:

```
1130 FOR I=1 TO LEN(C$)
1140 IF VAL(MID$(C$,I,1))=0 THEN L=L+1
1150 IF VAL(LEFT$(C$,I))<>0 THEN I=LEN(C$)
1160 NEXT I
1170 C$=RIGHT$(C$,LEN(C$)-L)
1180 IF S=1 THEN C$="-"+C$
```

Schritt 7: Ausgabe des Ergebnisses C$.

Nach Ausgabe von C$ mit einem PRINT-Befehl wird der Inhalt aller vom Programm verwendeten Variablen gelöscht, um das Programm auf eine neue Eingabe vorzubereiten:

```
1190 PRINT:PRINT"ERGEBNIS= ";C$:PRINT
1200 A$="":B$="":C$="":CH$="":CN$=""
1205 AH=0:AN=0:BH=0:BN=0:F=0:S=0:X=0:Y=0
1210 GOTO 20
1220 END
```

Das vollständige Programm zur Subtraktion in Parallelarithmetik sieht dann wie folgt aus:

```
10 PRINT"*** SUBTRAKTION IN PARALLELARITHMETIK ***":PRINT
15 ZERO$="0000000000000000"
20 INPUT A$,B$
25 IF VAL(A$)=VAL(B$) THEN C$="0":GOTO 1190
30 IF VAL(A$)>VAL(B$) GOTO 1000
40 X$=A$:A$=B$:B$=X$
50 S=1
1000 X=LEN(A$):Y=LEN(B$)
1002 IF X>Y THEN F=X/2:GOTO 1006
1004 F=Y/2
1006 IF F>INT(F) THEN F=INT(F)+1
1010 IF X<=F THEN AH=0:AN=VAL(A$):GOTO 1040
1020 AH=VAL(LEFT$(A$,X-F))
1030 AN=VAL(RIGHT$(A$,F))
1040 IF Y<=F THEN BH=0:BN=VAL(B$):GOTO 1070
1050 BH=VAL(LEFT$(B$,Y-F))
1060 BN=VAL(RIGHT$(B$,F))
1070 IF AN>=BN GOTO 1100
1080 AN=AN+10↑F
1090 AH=AH-1
1100 CN$=STR$(INT(AN-BN))
1101 CN$=MID$(CN$,2,LEN(CN$)-1)
1102 CN$=LEFT$(ZERO$,F-LEN(CN$))+CN$
1110 CH$=STR$(INT(AH-BH))
1111 CH$=MID$(CH$,2,LEN(CH$)-1)
1120 C$=CH$+CN$
1130 FOR I=1 TO LEN(C$)
1140 IF VAL(MID$(C$,I,1))=0 THEN L=L+1
1150 IF VAL(LEFT$(C$,I))<>0 THEN I=LEN(C$)
1160 NEXT I
1170 C$=RIGHT$(C$,LEN(C$)-L)
1180 IF S=1 THEN C$="-"+C$
1190 PRINT:PRINT"ERGEBNIS= ";C$:PRINT
1200 A$="":B$="":C$="":CH$="":CN$=""
1205 AH=0:AN=0:BH=0:BN=0:F=0:S=0:X=0:Y=0
1210 GOTO 20
1220 END
```

Hier die Wiedergabe einiger Programmläufe:

```
*** SUBTRAKTION IN PARALLELARITHMETIK ***

? 123456789012
?? 57943572

ERGEBNIS= 123398845440

? 1234567890123456
?? 57943572

ERGEBNIS= 1234567832179884

? 9999999999999999
?? 1234567890

ERGEBNIS= 9999998765432109
```

Sie haben jetzt zwei Verfahren der Subtraktion von mehr als 9stelligen Zahlen kennengelernt. Das erste Verfahren verwendet numerische Zeichenketten, das zweite Parallelarithmetik. Durch Vergleich der Ergebnisse können Sie sehen, daß beide Verfahren gleich gut arbeiten.

MULTIPLIKATIONEN

Bei der Multiplikation kann eine Begrenzung der Zahlendarstellung auf 9 Stellen leicht überschritten werden, da ein Produkt sehr groß werden kann, auch wenn Multiplikator und Multiplikand klein sind. Die Beschränkung der Stellenzahl führt dazu, daß Produkte mit mehr als 9 Stellen nur in Exponentialdarstellung wiedergegeben werden. Sie können diese Beschränkung durch ein Programm umgehen, das Produkte mit einer Genauigkeit von mehr als 9 Stellen auf dem Bildschirm ausgibt. Die Wiedergabe von mehr als 9stelligen Produkten ohne Exponentialdarstellung wird am leichtesten durch Multiplikation in Parallelarithmetik erreicht. Das folgende Programm und seine Beschreibung werden Sie in die Lage versetzen, Produkte mit einer Länge bis zu 16 Stellen ohne Exponentialdarstellung wiederzugeben.

MULTIPLIKATION IN PARALLELARITHMETIK

Mit praktisch den gleichen Schritten wie bei der Addition und Subtraktion in Parallelarithmetik werden Multiplikand und Multiplikator in kleinere Zahlenfelder geteilt, alle Zahlenfelder einer Multiplikation unterworfen, und aus der Addition der Teilprodukte das Endprodukt gebildet, das aus eins bis sechzehn Stellen bestehen kann.

Die Schritte der Multiplikation in Parallelarithmetik sind wie folgt:

1. Eingabe von Multiplikand und Multiplikator als zwei positive numerische Zeichenketten.
2. Teilung der Zeichenketten in höherwertige und niederwertige Zahlenfelder.
3. Multiplikation entsprechend der Zahlenfelder.
4. Addition der Teilprodukte zur Bildung des Ergebnisses.
5. Ergebnis-Darstellung.

Schritt 1: Eingabe von Multiplikand und Multiplikator.

Der Multiplikand wird als positive numerische Zeichenkette A$ eingegeben, der Multiplikator entsprechend als B$. Wie in den vorangegangenen Programmen kann durch Eingabe der Zahlen als Zeichenketten die im CBM-Computer vorhandene Beschränkung auf neunstellige Zahlen umgangen werden.

Das vorliegende Programm beschränkt die Länge eines Produktes auf 16 Stellen. Da die maximale Stellenzahl des Produkts gleich der Summe der Stellenzahlen von Multiplikand und Multiplikator ist, dürfen beide eingegebenen Zahlen zusammen höchstens 16 Stellen haben. Die Eingabe längerer Zahlen erfordert einige Programmänderungen, die nicht hier besprochen werden; Sie werden jedoch in der Lage sein, derartige Änderungen selbst auszuführen. Für unser Programm ergibt sich:

(Länge von A$)+(Länge von B$) ≤ 16

Beispiele:
$12 + 4 \leq 16$
$2 + 3 \leq 16$
$8 + 8 < 16$

Das Multiplikationsprogramm kann die beiden größtmöglichen 8stelligen Zahlen: 99999999 und 99999999 zu einem 16stelligen Produkt verarbeiten:

```
      99999999—      8 Stellen
    ×99999999—   +   8 Stellen
9999999800000001—    16 Stellen
```

Multiplikand und Multiplikator werden als positive numerische Zeichenketten A$ und B$ eingegeben:

```
10 PRINT"□*** MULTIPLIKATION IN PARALLELARITHMETIK ***":PRINT
20 INPUT A$,B$

RUN
*** MULTIPLIKATION IN PARALLELARITHMETIK ***

?99999999
??99999999
```

Schritt 2: Teilung der eingegebenen Zeichenketten in zwei Felder.

Die Feldlänge F bestimmt die Länge der Felder, in die A$ und B$ aufgeteilt werden. Die Bestimmung von F erfolgt in Zeilen 1002 und 1006 und wurde bereits unter "Addition in Parallelarithmetik" erklärt:

```
1000 X=LEN(A$):Y=LEN(B$)
     X=8        Y=8

1002 IF X>Y THEN F=X/2:GOTO 1008
1004 F=Y/2
     F=8/2
     F=4

1006 IF F>INT(F)THEN F=INT(F)+1
```

Sobald F festliegt, teilt das Programm die eingegebenen Zahlen in ein Feld mit den höherwertigen Stellen (H) und ein Feld mit den niederwertigen Stellen (N). In Zeilen 1010 bis 1060 erfolgt die Teilung in die Felder AH, AN und BH, BN. Diese Befehlsfolge wurde bereits in "Addition in Parallelarithmetik" verwendet:

```
1010 IF X<=F THEN AH=0:AN=VAL(A$):GOTO 1040
1020 AH=VAL(LEFT$(A$,X-F))
1030 AN=VAL(RIGHT$(A$,F))
1040 IF Y<=F THEN BH=0:BN=VAL(B$):GOTO 1070
1050 BH=VAL(LEFT$(B$,Y-F))
1060 BN=VAL(RIGHT$(B$,F))
```

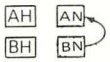
Ein Zahlenbeispiel zeigt das Ergebnis der Teilung von A$ und B$:

Schritt 3: Ausmultiplizieren von AH, AN, BH und BN.

Beim Ausmultiplizieren des Ausdrucks (AH + AN) * (BH + BL) entstehen die vier Produkte P1$, P2$, P3$ und P4$. Die Multiplikationen von A$ und B$ erfolgt also in vier Schritten, die in folgendem Schema dargestellt sind:

```
 AH  AN
×BH  BN
```

Zuerst wird BN mit AN multipliziert:

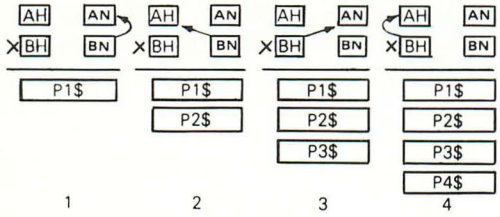


Dann folgt die Multiplikation von BN mit AH:

Danach folgen die Multiplikationen mit BH, zuerst BH * AN:

und schließlich die Multiplikation BH * AH:

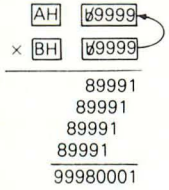
Hier die Zusammenstellung der vier Schritte und ihrer Produkte:



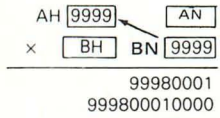
Lassen Sie uns Schritt für Schritt die Multiplikation durchgehen, indem wir für AH, AN, BH und BN die Zahlenwerte aus unserem Beispiel benutzen:

AH [ø9999] AN [ø9999]
BH [ø9999] BN [ø9999]

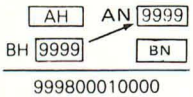
Die erste Multiplikation ist BN mal AN:



Die zweite Multiplikation ist BN mal AH:



Beachten Sie, daß Produkt P2$ nicht direkt unter Produkt P1$ steht, sondern um vier Stellen nach links verschoben, wie Sie es von der Anordnung der Teilprodukte bei Rechnungen mit Papier und Bleistift kennen. Die dritte Multiplikation sieht wie folgt aus:



Die vierte und letzte Multiplikation ist folgende:

```
AH [9999]  [AN]
BH [9999]  [BN]
-------------------
9998000100000000
```

Da in unserem Beispiel alle Felder aus dem Zahlenwert 9999 bestehen, geben alle Multiplikationen AN * BH, usw. das gleiche Produkt 99980001. Die stellenrichtige Verschiebung der Teilprodukte vor ihrer anschließenden Addition erfolgt im Programm durch Umwandlung der Produkte in Zeichenketten und das Anfügen der erforderlichen Nullen durch Addition einer aus Nullen bestehenden Zeichenkette F$. Zeilen 1070 bis 1100 führen diese Verschiebung durch:

```
1070 P1$=STR$(BN*AN)
1080 P2$=STR$(BN*AH)+F$
1090 P3$=STR$(BH*AN)+F$
1100 P4$=STR$(BH*AH)+F$+F$
```

Ohne stellenrichtige Verschiebung wäre die Berechnung des Resultats falsch:

```
P1 [99980001]
P2 [99980001]   falsch
P3 [99980001]
P4 [99980001]
-------------
   [        ]
```

Die zeilenrichtige Verschiebung der Teilprodukte ergibt folgendes Muster:

```
        [99980001]
    [99980001] 0000
    [99980001] 0000   richtig
[99980001] 00000000
-------------------
[                  ]
```

Die Zahl der an jedes Teilprodukt anzufügenden Nullen ist in der Zeichenkette F$ enthalten. Die Feldlänge F bestimmt die Anzahl der Nullen in F$. Zur Bildung von F$ wird die Nullen-Zeichenkette ZERO$ benutzt:

```
40 ZERO$="0000000000000000"
1008 F$=LEFT$(ZERO$,F)
     F$=LEFT$(ZERO$,4)
     F$=[0][0][0][0][0][0][0][0][0]
     F$="0000"
```

Nach stellenrichtiger Verschiebung der Teilprodukte in Zeilen 1070 bis 1100 erfolgt die Addition:

```
  AH [b̸][9][9][9][9]    AN [b̸][9][9][9][9]
× BH [b̸][9][9][9][9]    BN [b̸][9][9][9][9]
-------------------------------------------
            [99980001]   P1
        [999800010000]   P2
        [999800010000]   P3
    [9998000100000000]   P4
              F$  F$
```

Am Ende von Schritt 3 hat das Programm folgendes Aussehen:

```
20 INPUT A$,B$                                          Einlesen von A$, B$
30 IF VAL(A$)=0 OR VAL(B$)=0 THEN C$="0":GOTO 1190      Abfrage nach A$= 0
40 ZERO$="0000000000000000"                             und B$ = 0
1000 X=LEN(A$):Y=LEN(B$)
1002 IF X>Y THEN F=X/2:GOTO 1008                        Bestimmung der
1004 F=Y/2                                              Feldlänge F
1006 IF F>INT(F) THEN F=INT(F)+1
1008 F$=LEFT$(ZERO$,F)
1010 IF X<=F THEN AH=0:AN=VAL(A$):GOTO 1040
1020 AH=VAL(LEFT$(A$,X-F))
1030 AN=VAL(RIGHT$(A$,F))                               Aufteilung von A$,
1040 IF Y<=F THEN BH=0:BN=VAL(B$):GOTO 1070             B$ in je zwei Felder
1050 BH=VAL(LEFT$(B$,Y-F))
1060 BN=VAL(RIGHT$(B$,F))
1070 P1$=STR$(BN*AN)
1080 P2$=STR$(BN*AH)+F$                                 Multiplikation der
1090 P3$=STR$(BH*AN)+F$                                 Felder von A$, B$
1100 P4$=STR$(BH*AH)+F$+F$
```

Schritt 4: Addition der vier Teilprodukte P1$, P2$, P3$ und P4$.
Dies ist der schwierigste Teil der "Multiplikation in Parallelarithmetik", da zwischen dem Hauptprogramm und einem der Addition dienenden Unterprogramm fortlaufend Parameter auszutauschen sind. Wir werden aus dem Programm "Addition mit numerischen Zeichenketten" den in Schritt 5 beschriebenen Teil als Unterprogramm verwenden, um die Teilprodukte zu addieren. Wir bilden aus dem genannten Additionsprogramm nach Änderung der Zeilennummerierung folgendes Unterprogramm:

```
2000 REM UNTERPROGRAMM ZUR ADDITION
2010 BLANK$="                "
2020 X=LEN(A$):Y=LEN(B$)
2030 IF X<Y THEN A$=LEFT$(BLANK$,Y-X)+A$
2040 IF X>Y THEN B$=LEFT$(BLANK$,X-Y)+B$
2050 D=0:N=1:C$=""
2060 FOR I=LEN(A$) TO 1 STEP -1
2070 A=VAL(MID$(A$,I,1))
2080 A=A+D:D=0
2090 B=VAL(MID$(B$,I,1))
2100 C=A+B
2110 IF C>=10 THEN D=1
2120 IF D=1 AND I=1 THEN N=2
2130 C$=RIGHT$(STR$(C),N)+C$
2140 NEXT I
```

Vor Aufruf des Unterprogramms müssen die Summanden A$ und B$ mitgeteilt werden. Für die Addition der Teilprodukte P1$ und P2$ erfolgt diese Mitteilung in Zeile 1110:

```
1110 A$=P1$:B$=P2$
```

A$ | 9 | 9 | 9 | 8 | 0 | 0 | 0 | 1 |

B$ | 9 | 9 | 9 | 8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

Beachten Sie, daß die Variablen A$, B$ der Eingabe beliebiger Summanden dienen. Es können jeweils nur zwei Summanden übergeben werden, da das Unterprogramm nur zwei Zahlen gleichzeitig addieren kann.

Nach Übergabe der Werte von P1$ und P2$ an A$ und B$ wird das Unterprogramm wie folgt aufgerufen:

```
1120 GOSUB 2000
```

Vor der Addition werden A$ und B$ rechtsbündig verschoben und durch Auffüllen mit Leerstellen aus dem Vorrat der Zeichenkette BLANK$ längengleich gemacht (Zeilen 2010 bis 2040):

```
2010 BLANK$="                    "
2020 X=LEN(A$):Y=LEN(B$)
2030 IF X<Y THEN A$=LEFT$(BLANK$,Y-X)+A$
2040 IF X>Y THEN B$=LEFT$(BLANK$,X-Y)+B$
```

Die stellenweise Addition der Zahlenwerte in A$ und B$ erfolgt in Zeilen 2050 bis 2140. Die Summe C wird danach in eine numerische Zeichenkette C$ zurückverwandelt. Eine vollständige Erklärung finden Sie in Schritt 3 des Programms "Addition mit numerischen Zeichenketten":

```
2050 D=0:N=1:C$=""
2060 FOR I=LEN(A$) TO 1 STEP-1
2070 A=VAL(MID$(A$,I,1))
2080 A=A+D:D=0
2090 B=VAL(MID$(B$,I,1))
2100 C=A+B
2110 IF C>=10 THEN D=1
2120 IF D=1 AND I=1 THEN N=2
2130 C$=RIGHT$(STR$(C),N)+C$
2140 NEXT I
```

Die Summe C$ wird in einer FOR-NEXT-Schleife verarbeitet, um alle führenden Leerstellen und Nullen abzuschneiden. Zeilen 3000 bis 3060 führen diese vom Programm "Subtraktion mit numerischen Zeichenketten" entnommene Prozedur durch:

```
3000 REM UNTERDRUECKEN FUEHRENDER NULLEN
3001 L=0
3010 FOR I=1 TO LEN(C$)
3020 IF VAL(MID$(C$,I,1))=0 THEN L=L+1
3030 IF VAL(LEFT$(C$,I))<>0 THEN I=LEN(C$)
3040 NEXT I
3050 C$=RIGHT$(C$,LEN(C$)-L)
3060 RETURN
```

C$, die Summe von P1$ und P2$, wird an das Hauptprogramm zurückgegeben und dort unter dem Variablennamen M1$ gespeichert:

```
1130 M1$=C$
```

Der Inhalt von C$ muß in M1$ gespeichert werden, da C$ vor Wiederbenutzung des Unterprogramms gelöscht wird.

Danach werden die Teilprodukte P3$ und P4$ an die Parameter A$ und B$ des Unterprogramms übergeben und das Unterprogramm aufgerufen:

```
1132 A$=P3$:B$=P4$:GOSUB 2000
```

A$		9	9	9	8	0	0	0	0	0	0	0
B$	9	9	9	8	0	0	0	1	0	0	0	0	0	0	0

Das Unterprogramm addiert P3$ und P4$ stellenrichtig, trennt alle führenden Leerstellen und Nullen ab und gibt die Summe C$ an das Hauptprogramm zurück, wo sie unter dem Variablennamen M2$ gespeichert wird:

```
1135 M2$=C$
```

Das Unterprogramm wird dann ein drittes Mal aufgerufen, um die Summanden M1$ und M2$ zu addieren und ihre Summe C$ zurückzumelden:

```
1140 H$=M1$:B$=M2$
```

A$ [9][9][9][8][9][9][9][0][0][1]

B$ [9][9][9][8][9][9][9][0][0][1][0][0][0]

```
1150 GOSUB 2000
```

Schritt 5: Ausgabe des Endergebnisses.

Nach der dritten Rückkehr aus dem Unterprogramm entspricht C$ dem Endergebnis, d.h. der Summe aller vier Teilprodukte. Der Befehl GOTO 20 läßt das Programm an seinen Anfang zurückspringen und erlaubt eine erneute Eingabe:

```
1190 PRINT:PRINT"ERGEBNIS= ";C$:PRINT:GOTO 20
1200 END
```

Die Schritte 1-5 fügen sich zu folgendem Programmfluß:

```
Schritt 1   10  PRINT"***MULTIPLIKATION IN PARALLELARITHMETIK***
            20
            30  Eingabe von Multiplikand und Multiplikator.
            40  Initiierung der Variablen.
            50

Schritt 2   1000
            1010
            1020
            1030  Berechnung von F. Aufteilung von Multiplikator und
            1040  Multiplikand in je zwei Felder
            1050
            1060

Schritt 3   1070
            1080  Ausmultiplizieren der Felder mit den Ergebnissen
            1090  P1$, P2$, P3$, P4$
            1100

Schritt 4   1110  Übergabe von P1$, P2$ an A$, B$
            1120  GOSUB 2000 ------------------> 2000-2140  Additionsprogramm
                                                   P1$ + P2$ → C$
                  <------------------------------ 3000-3060  Unterdrückung führender
            1130  Übergabe von C$ an M1$                     Nullen
            1140  Übergabe von P3$, P4$ an A$, B$
            1150  GOSUB 2000 ------------------> 2000-2140  Additionsprogramm
                                                   P3$ + P4$ → C$
                  <------------------------------ 3000-3060  Unterdrückung führender
            1160  Übergabe von C$ an M2$                     Nullen
            1170  Übergabe von M1$, M2$ an A$, B$
            1180  GOSUB 2000 ------------------> 2000-2140  Additionsprogramm
                                                   M1$ + M2$ → C$
                  <------------------------------ 3000-3060  Unterdrückung führender
                                                             Nullen
Schritt 5   1190  Prints C$
            1200  END
```

Und hier die Auflistung des vollständigen Programms:

```
10 PRINT"□**** MULTIPLIKATION IN PARALLELARITHMETIK ***":PRINT
20 INPUT A$,B$
30 IF VAL(A$)=0 OR VAL(B$)=0 THEN C$="0":GOTO 1190
40 ZERO$="0000000000000000"
1000 X=LEN(A$):Y=LEN(B$)
1002 IF X>Y THEN F=X/2:GOTO 1008
1004 F=Y/2
1006 IF F>INT(F) THEN F=INT(F)+1
1008 F$=LEFT$(ZERO$,F)
1010 IF X<=F THEN AH=0:AN=VAL(A$):GOTO 1040
1020 AH=VAL(LEFT$(A$,X-F))
1030 AN=VAL(RIGHT$(A$,F))
1040 IF Y<=F THEN BH=0:BN=VAL(B$):GOTO 1070
```

```
1050 BH=VAL(LEFT$(B$,Y-F))
1060 BN=VAL(RIGHT$(B$,F))
1070 P1$=STR$(BN*AN)
1080 P2$=STR$(BN*AH)+F$
1090 P3$=STR$(BH*AN)+F$
1100 P4$=STR$(BH*AH)+F$+F$
1110 A$=P1$:B$=P2$
1120 GOSUB 2000
1130 M1$=C$
1132 A$=P3$:B$=P4$:GOSUB 2000
1135 M2$=C$
1140 A$=M1$:B$=M2$
1150 GOSUB 2000
1190 PRINT:PRINT"ERGEBNIS= ";C$:PRINT:GOTO 20
1200 END
2000 REM UNTERPROGRAMM ZUR ADDITION
2010 BLANK$="                    "
2020 X=LEN(A$):Y=LEN(B$)
2030 IF X<Y THEN A$=LEFT$(BLANK$,Y-X)+A$
2040 IF X>Y THEN B$=LEFT$(BLANK$,X-Y)+B$
2050 D=0:N=1:C$=""
2060 FOR I=LEN(A$) TO 1 STEP -1
2070 A=VAL(MID$(A$,I,1))
2080 A=A+D:D=0
2090 B=VAL(MID$(B$,I,1))
2100 C=A+B
2110 IF C>=10 THEN D=1
2120 IF D=1 AND I=1 THEN N=2
2130 C$=RIGHT$(STR$(C),N)+C$
2140 NEXT I
3000 REM UNTERDRUECKEN FUEHRENDER NULLEN
3001 L=0
3010 FOR I=1 TO LEN(C$)
3020 IF VAL(MID$(C$,I,1))=0 THEN L=L+1
3030 IF VAL(LEFT$(C$,I))<>0 THEN I=LEN(C$)
3040 NEXT I
3050 C$=RIGHT$(C$,LEN(C$)-L)
3060 RETURN
```

und das Beispiel eines Programmlaufs:

```
**** MULTIPLIKATION IN PARALLELARITHMETIK ***

? 99999999
?? 99999999

ERGEBNIS= 9999999800000001
```

Computer-Graphiken sind ein eigenes Gebiet. Ganze Bücher wurden diesem Gebiet gewidmet. Die nachfolgende Behandlung von Computer-Graphiken hat sich notwendigerweise kurz zu halten.

CBM-Computer verfügen über einen Standard-Satz von 64 graphischen Symbolen. Ein Alternativ-Satz mit jedoch nur wenigen graphischen Symbolen steht nach Eingabe des Befehls POKE 59468,14 zur Verfügung. Bei CBM 8000-Computern steht eine besondere Funktion zur Edition graphischer Symbole zur Verfügung (siehe Kapitel 8):

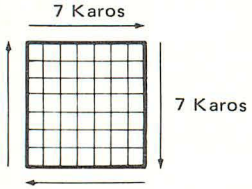
```
100 PRINT CHR$(142): REM UMSCHALTUNG AUF GRAPHISCHE SYMBOLE
```

Zur Eingabe graphischer Symbole muß die Umschalttaste für Großbuchstaben (SHIFT) betätigt werden.

Viele der graphischen Symbole werden auf den folgenden Seiten erwähnt und wiedergegeben. Die Tasten, Bezeichnungen und Wiedergaben graphischer Symbole finden Sie in Tabelle 1-1 oder im Anhang A.

EINGABE VON COMPUTER-GRAPHIKEN

Graphische Darstellungen in direktem Dialog mit dem Computer erfordern keine Zeilennummern, keine PRINT-Befehle und keine Anführungszeichen. Bei direktem Dialog kann der Kursor frei über den gesamten Bildschirm bewegt werden, ohne daß nach jeder Richtungsänderung die Taste RETURN zu drücken wäre. Ein Beispiel: Mit einer durchgehenden Kursor-Bewegung wurde ein Viereck dargestellt, indem der Kursor von seiner Ausgangsstellung nach rechts, dann von oben nach unten, dann von rechts nach links und schließlich zurück in die Ausgangsstellung bewegt wurde. Es waren keine Zeilennummern, Programmbefehle oder Wagenrückläufe (RETURN) erforderlich:

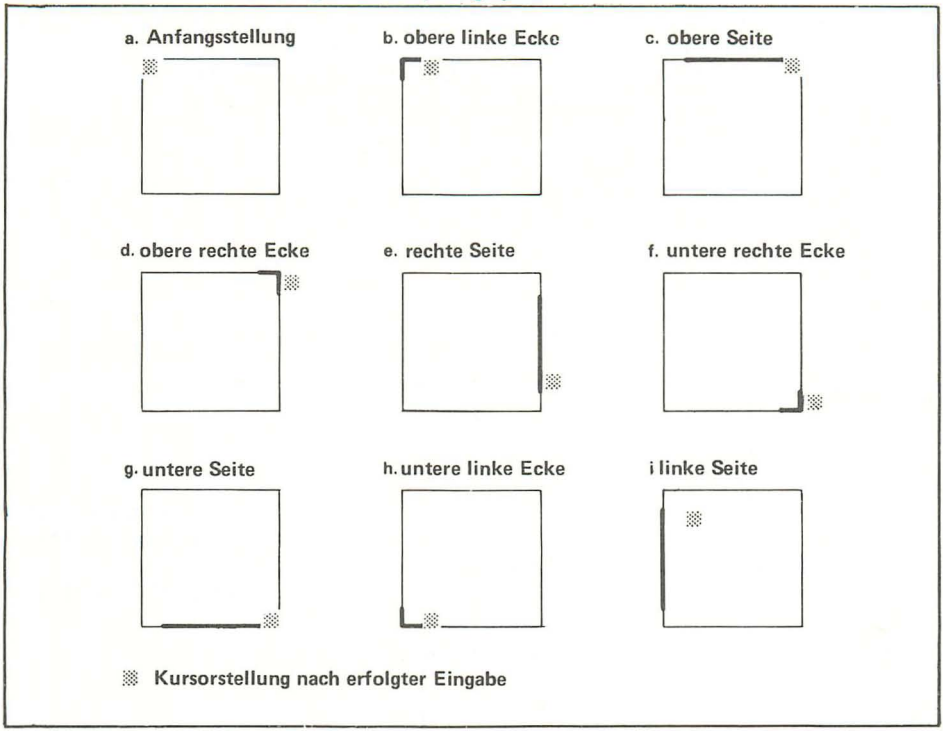


Wir werden das oben gezeigte Viereck als einfachen graphischen Entwurf zur Erläuterung der Grundlagen von Computer-Graphiken benutzen. Obwohl einfach in seiner Gestalt, nutzt dieses Viereck bereits alle graphischen Darstellungstechniken eines CBM-Computers aus.

Die Darstellung eines Vierecks mit 7 x 7 Karos umfaßt neun Schritte:

Schritt 1: Bewegung des Kursors in den Anfangszustand (HOME). Bei Betätigung der Taste HOME wird der Kursor in seine Ausgangsstellung, d.h. in die linke obere Ecke des Bildschirms gebracht. Diese Ausgangsstellung ist zugleich die linke obere Ecke des Vierecks (Figur5 - 2 a)

Figur 5 - 2.Phasen der Entstehung einer Computergraphik



Schritt 2: Darstellung der oberen linken Ecke des Vierecks. Die Tastatur enthält das graphische Symbol einer oberen linken Ecke (Figur 5 - 2 b)

Schritt 3: Darstellung der oberen Begrenzungslinie des Vierecks. Da wir für die obere rechte Ecke das Eckensymbol verwenden wollen, drücken wir fünfmal die Taste für eine obere horizontale Linie (Figur 5 - 2c)

Schritt 4: Darstellung der oberen rechten Ecke des Vierecks. Das Tastenfeld ermöglicht die Ausgabe des entsprechenden Eckensymbols (Figur 5 - 2 d)

Schritt 5: Darstellung der vertikalen rechten Begrenzungslinie des Vierecks. Mit Rücksicht auf ein noch folgendes Eckensymbol wird fünfmal das graphische Symbol für eine rechte vertikale Linie eingegeben.

Wir wissen alle, wie dieser Teil des Vierecks aussehen sollte, aber hat sich vielleicht folgendes Bild ergeben?

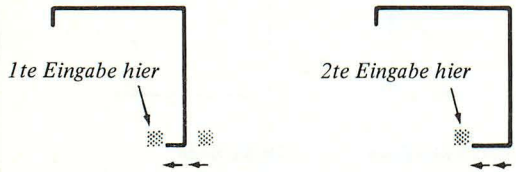
Der Kursor wird nach jeder Symboleingabe automatisch um eine Stelle nach rechts, das heißt in horizontaler Richtung verschoben. Für eine vertikale Darstellung muß der Kursor sowohl in horizontaler wie vertikaler Richtung korrigiert werden, um automatische Kursor-Bewegungen zu kompensieren.

Zur Darstellung der rechten vertikalen Seite des Vierecks wiederholen Sie fünfmal folgende Tasteneingaben: CURSOR DOWN, CURSOR LEFT (s. Tabelle 1-1) und die Taste für eine rechte vertikale Linie (s. Tabelle 1-1). Danach sollte die rechte Seite des Vierecks dargestellt sein (Figur 5 - 2 e)

Schritt 6: Darstellung der rechten unteren Ecke des Vierecks. Hierfür wird die Taste für ein rechtes unteres Eckensymbol benutzt. Bevor Sie diese Taste drücken, achten Sie jedoch auf die Stellung des Kursors. Von seiner Position in Figur 5 - 2e sollte der Kursor durch Benutzung der Tasten CURSOR DOWN und CURSOR LEFT in die Ausgangsstellung für eine Eckendarstellung gebracht werden. Danach geben Sie das Eckensymbol ein (Figur 5 - 2 f)

Schritt 7: Darstellung der unteren Begrenzungslinie des Vierecks. Wegen des noch folgenden linken Eckensymbols benötigen wir nur fünf Eingaben mit der Taste für untere horizontale Linien (Figur 5 - 2 g)

Die Eingabe dieser Linie kann von rechts nach links erfolgen. Vor jeder Symboleingabe muß die Taste CURSOR LEFT zweimal gedrückt werden, um den Kursor in die richtige Position zu bringen:



In einer zweiten und vielleicht natürlicheren Art kann die untere Linie auch von links nach rechts gezeichnet werden. Hierzu wird der Kursor durch sechsmaliges Betätigen der Taste CURSOR LEFT nach linke verschoben, bis er eine Stelle vor der linken Begrenzungslinie des Vierecks steht. Danach geben Sie die untere horizontale Linie durch fünfmaliges Drücken der entsprechenden graphischen Taste ein:

Schritt 8: Eingabe der linken unteren Ecke des Vierecks. Je nach der Art, die Sie zur Eingabe der unteren Begrenzungslinie gewählt haben, müssen Sie den Kursor durch zweimaliges oder sechsmaliges Drücken der Taste CURSOR LEFT in die untere linke Ecke verschieben und dann die Taste für das graphische Symbol einer unteren linken Ecke drücken (Figur 5 - 2 h Tabelle 1-1).

Schritt 9: Vervollständigung des Vierecks durch die linke Begrenzungslinie. Es sollte Ihnen jetzt keine Schwierigkeit bereiten, fünf linke vertikale Linien einzugeben (Figur 5 - 2 i. Vor jeder Eingabe muß der Kursor mit den Tasten CURSOR LEFT und CURSOR UP positioniert werden.

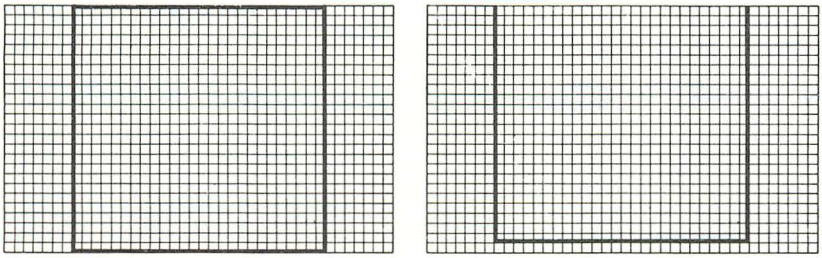
REPRODUKTION VON COMPUTER-GRAPHIKEN

Jede auf dem Bildschirm entworfene Computer-Graphik geht verloren, wenn Sie den Befehl NEW eingeben oder den Computer ausschalten, es sei denn, Sie haben die Graphik zuvor in ein Programm verwandelt. Sie können jeden graphischen Entwurf auf dem Bildschirm einfach dadurch in ein Programm umwandeln, daß Sie jede Zeile auf dem Bildschirm zu einer Zeichenkette erklären, so daß sie als Teil eines Programms gespeichert und wieder ausgegeben werden kann.

Bringen Sie nach Darstellung des Vierecks den Kursor mit der Taste HOME in seine Ausgangsstellung. Betätigen Sie jedoch nicht die Tasten CLEAR oder RETURN. Bei Benutzung der Taste CLEAR geht die Graphik für immer verloren. Bei Benutzung der Taste RETURN wird die Antwort "READY" quer durch das Viereck geschrieben, wie unten gezeigt:

Eine Graphik kann dadurch zerstört werden, daß Teile von ihr auf der obersten oder untersten Bildschirmzeile stehen. Erfolgt jetzt die Eingabe RETURN, dann schafft der Computer für die Ausgabe der Antwort READY Platz, indem er die gesamte Computergraphik um eine Zeile nach oben schiebt. Hierbei geht die Oberseite der Graphik verloren:

vor RETURN **nach RETURN**



Aus diesem Grunde sollten Graphiken nicht mehr als 39 Symbole in der Horizontalen und 24 Symbole in der Vertikalen enthalten. Bei CBM 8000-Computern sollten es nicht mehr als 79 Symbole in der Horizontalen sein.

Der nächste Schritt nach Rückkehr des Kursors in seine Ausgangsstellung ist, jede Zeile der Graphik so weit nach rechts zu schieben, daß eine Zeilennummer, das Fragezeichen (als Abkürzung für den PRINT-Befehl) und Anführungszeichen eingefügt werden können. Hierdurch wird jede Zeile der Graphik Teil eines Programms, das auf Kassette oder Diskette gespeichert werden kann.

Die Umwandlung einer Graphik in ein Programm wird am Beispiel des oben gezeichneten Quadrats demonstriert. Der Kursor sollte sich nach Betätigung der HOME-Taste in der linken oberen Ecke des Vierecks befinden (Figur 5 - 3 a) Drücken Sie jetzt die Taste INSERT fünfmal, so daß die obere Bildschirmzeile um 5 Stellen nach rechts verschoben wird (Figur 5 - 3 b) Jetzt ist genug Platz entstanden, um die Zeilennummer 100, ein Fragezeichen und die Anführungszeichen zur Eröffnung einer Zeichenkette einzutragen (Figur 5 - 3 c) Danach drücken Sie die Taste RETURN (Figur 5 - 3 d) Die obere Begrenzungslinie des Quadrats ist jetzt zu einem Programmbefehl geworden. Wiederholen Sie diesen Vorgang für jede Zeile, wobei Sie die Zeilennummern in Schritten von 100 erhöhen, bis das gesamte Quadrat in Programmbefehle umgewandelt ist (Figur 5 - 3 e,f)

Achten Sie darauf, die Zeilennummern in aufsteigender Ordnung zu vergeben, um eine Störung der Graphik bei der Wiedergabe zu vermeiden. Weiterhin ist wichtig zu wissen, daß Sie den Kursor an das Ende jeder Zeile der Graphik bewegen müssen, um dort das Anführungszeichen zum Abschluß einer Zeichenkette einzuge-

ben. Nach Eingabe der ersten Anführungszeichen drücken Sie einfach die Taste RETURN. Die anschließende Auflistung des Programms sollte folgendes Bild ergeben:

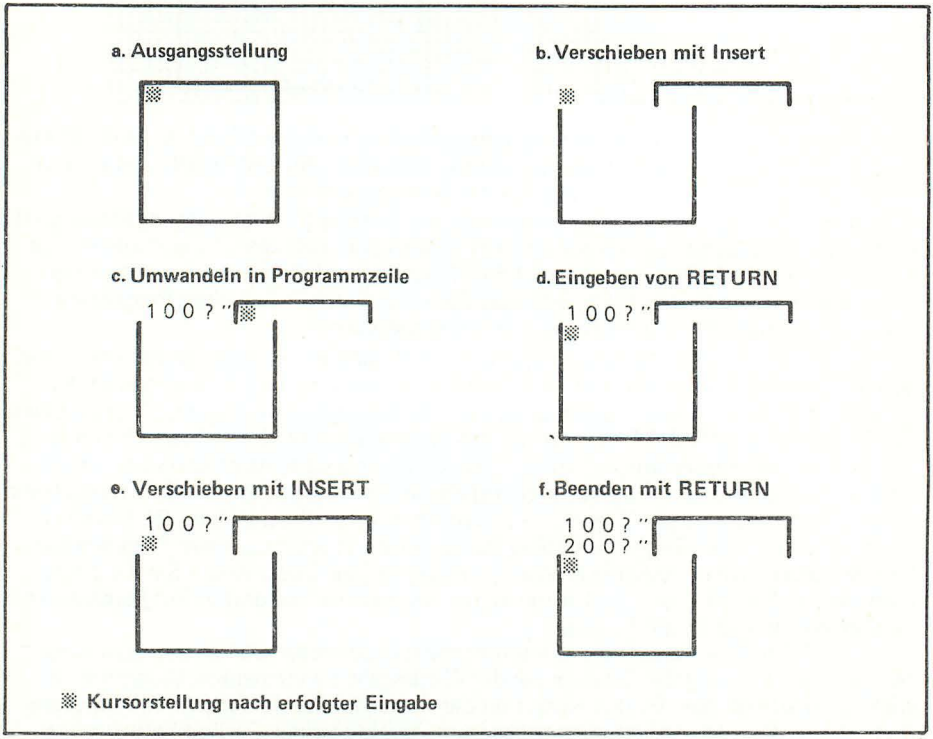
```
100 PRINT"┌──────┐
200 PRINT"│      │
300 PRINT"│      │
400 PRINT"│      │
500 PRINT"│      │
600 PRINT"│      │
700 PRINT"└──────┘
```

Sie können eine Graphik aber auch von Anfang an als Programm eingeben. Jede Zeile der Graphik wird dann als Teil eines PRINT-Befehls eingegeben:

```
100 ?"┌──────┐
200 ?"│      │
300 ?"│      │
```

Die Leerstelle unmittelbar rechts neben dem Fragezeichen wird bei der Wiedergabe der Graphik als Spalte 1 benutzt. Ihre Graphik erscheint daher bei der Wiedergabe auf dem Bildschirm um eine Stelle nach links verschoben.

Bei der PRINT-Eingabe einer Zeichenkette, die aus genau 40 Symbolen besteht, müssen Sie auch die Anführungszeichen am Ende der Zeichenkette einfügen und die Zeile mit einem Semikolon abschließen. Geben Sie das Semikolon nicht ein, dann wird eine zusätzliche Zeile auf dem Bildschirm dargestellt, da der Kursor nach Darstellung des vierzigsten Symbols in einer Zeile auf die folgende Zeile springt:



Figur 5 - 3. Phasen der Programmierung einer Computer - Graphik

Noch ein Hinweis zum Arbeiten mit Computer-Graphiken: Es ist ratsam, die Graphik oder das Diagramm zunächst auf ein Stück Papier zu zeichnen. Notieren Sie auf einem Stück karierten Papier eine Fläche, die 40 Karos (bei CBM 8000-Computern: 80 Karos) breit und 25 Karos hoch ist. Vergessen Sie nicht den Platz für die Zeilennummer, wenn Sie die Graphik in ein Programm umwandeln wollen. Sobald die Graphik entworfen ist, übertragen Sie sie vom Papier auf den Bildschirm.

BELEBTE COMPUTER-GRAPHIKEN

Jede Darstellung auf dem Bildschirm, ob Zahl, Wort oder graphischer Entwurf, kann zeitabhängig verändert werden, indem Sie programmgesteuert über den Bildschirm bewegt wird, aufblinkt oder verlangsamt dargestellt wird. Diese zeitabhängigen Veränderungen können in fast jeder Kombination programmiert werden.

Zur Demonstration der Belebung einer Computer-Graphik werden wir mit dem Quadrat aus dem vorigen Abschnitt beginnen. Einem Betrachter soll das Quadrat nicht schlagartig, sondern verzögert auf dem Bildschirm dargestellt werden, so daß er die Entstehung des Quadrats aus seinen Elementen auf dem Bildschirm beobachten kann.

Das Programm zur zeitabhängigen Darstellung des Quadrats ist sehr verschieden vom vorangehenden Programm, da das Programm nicht ganze Bildschirmzeilen speichert, sondern jedes Segment des Quadrats über BASIC-Befehle ausgibt.

ZEITVERZÖGERTE BILDSCHIRMAUSGABE

Das Programm zur Belebung der Computer-Graphik verzögert die Kursor-Bewegung derart, daß sich das Quadrat langsam auf dem Bildschirm entwickelt. Die Darstellung beginnt in der linken oberen Ecke des Bildschirms und schreitet im Uhrzeigersinn weiter:

Der erste Schritt ist wie immer die Löschung des Bildschirms. Hierdurch wird außerdem der Kursor in seine Ausgangsstellung gebracht:

```
5 PRINT"□";
```

Der zweite Schritt ist die Darstellung der linken oberen Ecke. Stellen Sie aber nicht die ganze Oberkante des Quadrats dar, wie es im vorigen Programm erfolgte, sondern nur die Ecke:

```
10 PRINT"┌";
```

Um beobachten zu können, wie jedes Element des Quadrats auf dem Bildschirm erscheint, ist eine Verzögerung der Befehlsausführung erforderlich. Hierzu wird vor jedem PRINT-Befehl eine Programmschleife eingefügt, die eine Verzögerung der Programmausführung bewirkt:

```
100 FOR J=1 TO 100:NEXT J:RETURN
```

Die FOR-NEXT-Schleife stellt ein Unterprogramm dar, das den Computer zwingt, von 1 bis 100 zu zählen. Durch Veränderung der oberen Zählgrenze von J (die Zahl hinter TO) kann die eintretende Verzögerung beeinflußt werden. Je größer

die Zahl für die obere Zählgrenze, desto länger die Verzögerung zwischen der Darstellung einzelner Elemente des Quadrats.

Zur Belebung unserer Computer-Graphik müssen wir daher diese Verzögerungsschleife nach jeder Darstellung eines Bildelements einfügen. Da zwischen allen Bildelementen die gleiche Zeitverzögerung liegt, bilden wir aus der Verzögerungsschleife ein Unterprogramm. In unserem Beispiel: Nach Darstellung der oberen linken Ecke des Quadrats erfolgt ein Sprung zum Unterprogramm für Zeitverzögerungen in Zeile 100.

PROGRAMMIERUNG DER SYMBOLAUSGABEN

Im dritten Schritt soll die obere Begrenzungslinie des Quadrats ausgegeben werden. Statt für jedes Element dieser Linie einen eigenen PRINT-Befehl vorzusehen, werden wir eine FOR-NEXT-Schleife verwenden:

```
15 FOR I=1 TO 5:PRINT"─";:GOSUB 100:NEXT I
```

Bei jedem Durchlauf der Schleife FOR-NEXT in Zeile 15 erfolgt mit GOSUB 100 der Sprung in das Unterprogramm in Zeile 100, um die Ausgabe des nächsten Bildelements zu verzögern. Wenn der Computer das Quadrat langsam darstellen soll, dann muß die Zeitverzögerung nach jedem Bildsymbol eingefügt werden. Es würde nichts helfen, folgendes zu programmieren:

```
15 PRINT"─────";:GOSUB 100 ←— unzulässig
```

da dann die obere Begrenzungslinie des Quadrats schlagartig dargestellt wird.

Zum Abschluß der oberen Begrenzungslinie wird noch die obere rechte Ecke ausgegeben. Nach diesem PRINT-Befehl wird wieder eine Verzögerung eingefügt:

```
20 PRINT"┐";:GOSUB 100
```

Bis jetzt hat das Programm folgendes Aussehen:

```
5 PRINT"┌";
10 PRINT"┌";:GOSUB 100
15 FOR I=1 TO 5:PRINT"─";:GOSUB 100:NEXT I
20 PRINT"┐";:GOSUB 100
30 END
100 FOR I=1 TO 100:NEXT J:RETURN
```

Lassen Sie das Programm ablaufen. Die Darstellung auf dem Bildschirm sollte fortschreiten von ┌ bis ┌ ┐.

Hoffentlich war es der Fall. Wenn nicht, prüfen Sie, ob alle PRINT-Befehle in diesem Programm mit einem Semikolon (;) abgeschlossen wurden. Das Semikolon verkettet bei der Ausgabe graphische Zeichenketten. Dadurch wird das Eckensymbol "┌" und die obere Begrenzungslinie "──────────" auf der gleichen Bildschirmzeile verkettet. Ohne das Semikolon führt der CBM-Computer nach jedem PRINT-Befehl einen Wagenrücklauf aus, so daß die obere Begrenzungslinie des Quadrats wie folgt dargestellt wird:

```
┌
─
─
─
─
─
┐
```

Für die anderen drei Seiten gilt das gleiche Verfahren. Nach Zeile 20 beginnt die nächste Folge von PRINT-Befehlen, mit denen die rechte vertikale Begrenzungslinie des Quadrats dargestellt wird. Vergessen Sie in der FOR-NEXT-Schleife nicht

die Kontrollbefehle, mit denen die automatischen Rechtsbewegungen des Kursors kompensiert werden.

Die PRINT-Befehle in den FOR-NEXT-Schleifen für die Darstellung der rechten, unteren und linken Begrenzungslinie des Quadrats haben folgendes Aussehen:

PRINT" ▮▮" Rechte Seite Tasten:" <rechte vertikale Linie><Kursor links><Kursor n. unten>"

PRINT"_▮▮▮" Unterkante Tasten:" <untere Horizontale Linie><Kursor links><Kursor links>"

PRINT"▮▮" Linke Seite Tasten:" <linke vertikale Linie><Kursor links><Kursor n. oben>"

Die Auflistung des vollständigen Programms ergibt folgendes:

```
5 PRINT"┐";
10 PRINT"┌";:GOSUB 100
15 FOR I=1TO 5:PRINT"─";:GOSUB 100:NEXT I
20 PRINT"┐";:GOSUB 100
25 FOR I=1TO 5:PRINT" ▮▮";:GOSUB 100:NEXT I
30 PRINT"┘";:GOSUB 100
35 FOR I=1TO 5:PRINT"_▮▮▮";:GOSUB 100:NEXT I
40 PRINT"└";:GOSUB 100
45 FOR I=1TO 5:PRINT"▮▮";:GOSUB 100:NEXT I
50 END
100 FOR J=1 TO 10:NEXT J:RETURN
```

Versuchen Sie jetzt einen Probelauf. Hat Ihr Quadrat etwa folgendes Aussehen?:

```
┌──────┐
READY.  │
    │   │
    │   │
    │   │
  └─┘ ────
```

Wenn diese Wiedergabe eines Quadrats erfolgt, dann wurden einige Kontrollbefehle für den Kursor vergessen. Der Computer hat sich genau an das Programm gehalten, wo aber ist der Fehler? Schauen Sie sich das Programm noch einmal genau an. Kontrollbefehle für den Kursor sind in allen FOR-NEXT-Schleifen für die Begrenzungslinie des Quadrats enthalten. Jetzt sehen Sie auf den Bildschirm. Das Problem tritt nicht bei den Begrenzungslinien, sondern bei den Ecken auf. Prüfen Sie die Befehle in den Zeilen 20, 30 und 40. Wir haben offenbar die Kontrollbefehle des Kursors nach Darstellung der Eckensymbole vergessen. Fügen Sie die Änderungen ein, das Programm sollte dann wie folgt aussehen:

```
5 PRINT"┐";
10 PRINT"┌";:GOSUB 100
15 FOR I=1 TO 5:PRINT"─";:GOSUB 100:NEXT I
20 PRINT"┐";:GOSUB 100
25 FOR I=1 TO 5:PRINT" ▮▮";:GOSUB 100:NEXT I
30 PRINT"┘";:GOSUB 100
35 FOR I=1 TO 5:PRINT"_▮▮▮";:GOSUB 100:NEXT I
40 PRINT"└";:GOSUB 100
45 FOR I=1 TO 5:PRINT"▮▮";:GOSUB 100:NEXT I
50 END
100 FOR I=1 TO 100:NEXT J:RETURN
```

Versuchen Sie jetzt einen weiteren Probelauf. Die Graphik sollte dann wie folgt aussehen:

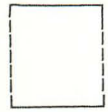
Sie werden beobachtet haben, wie der Computer das Quadrat langsam im Uhrzeigersinn zeichnet. Erinnern Sie sich, daß Sie die Darstellungsgeschwindigkeit durch Änderung der oberen Zählergrenze der Variablen J in der Verzögerungsschleife verändern können.

Ein letztes Problem: Wie kann vermieden werden, daß die READY-Nachricht das Quadrat zerstört?

Nach Zeichnung des Quadrats steht der Kursor auf Bildzeile 2; bei Programmende erfolgt die Ausgabe READY auf der folgenden Zeile, die sich innerhalb des Quadrats befindet. Wenn wir daher vor Programmende den Kursor unterhalb des Quadrats positionieren, wird die Nachricht READY unterhalb des Quadrats und nicht in das Quadrat geschrieben. Wir geben daher vor dem Endebefehl END mehrere Befehle CURSOR COWN ein:

```
50 PRINT"■■■■■■■■" :END
```

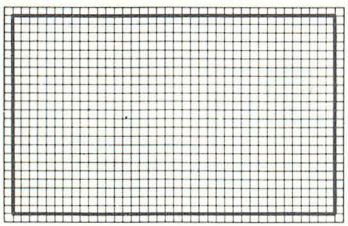
Hierdurch bewegt sich der Kursor unter das Quadrat und die Graphik wird nicht zerstört:



```
READY.
■
```

VERGRÖSSERUNG EINER RECHTECKDARSTELLUNG

Lassen Sie uns das kleine Quadrat, das wir oben zur Darstellung verwendet haben, derart vergrößern, daß es die Umrandung eines 40spaltigen Bildschirms bildet, wobei wir jedoch eine Zeile bzw. Spalte Abstand zum Bildschirmrand halten:



Besteht der Bildschirm aus 40 Spalten und 25 Zeilen, dann sollten die Seiten des Rechtecks aus 38 bzw. 23 Symbolen bestehen.

Mit nur wenigen Änderungen des Programms zur verzögerten Darstellung des Quadrats können wir ein vergrößertes Rechteck zeichnen. Das vorhergehende Programm benutzte FOR-NEXT-Schleifen, um eine Kette graphischer Symbole für jede Begrenzungslinie zu zeichnen. Zur Vergrößerung des Rechtecks ändern Sie für die horizontalen Begrenzungslinien die obere Zählgrenze der Variablen I auf 36, und für die vertikalen Begrenzungslinien auf 21, womit noch Platz bleibt für die Eckensymbole:

```
15 FOR I=1 TO 36:?"¯";
25 FOR I=1 TO 21:?" ▮▯";
35 FOR I=1 TO 36:?"_▮▮";
45 FOR I=1 TO 21:?"| ▮▯";
```

Horizontale Seiten

Vertikale Seiten

Machen Sie in Ihrem Programm diese Änderungen und versuchen Sie einen Probelauf.

Das war einfach. Da Sie aber jetzt eine Umrandung des Bildschirms programmiert haben, wird Zeile 50 zur Bewegung des Kursors unterhalb des Quadrats überflüssig. Da wir nicht einen leeren Bildschirm umranden wollen, benutzen wir jetzt Zeile 50, um den Kursor innerhalb der Umrandung zu positionieren, wo dann eine Eingabe erfolgen kann. Achten Sie darauf, den Kursor nicht unterhalb des Rechtecks zu positionieren, da sonst die Umrandung nach oben aus dem Bildschirm geschoben wird und verschwindet. Programmieren Sie irgendeine Darstellung innerhalb der Umrandung!

DIE ECHTZEIT-UHR IM CBM-COMPUTER

Eine weitere Ausstattung von CBM-Computern ist die Echtzeit-Uhr, die die Tageszeit im 24 Stunden-Zyklus nach Stunden, Minuten und Sekunden anzeigt. Die laufende Zeit kann über die Variablen TIME$ oder TI$ abgelesen werden.

STELLEN DER ECHTZEIT-UHR

Das Stellen der Uhr geschieht in folgender Form:

TIME$ = "hhmmss"

mit:	Stunde	hh	zwischen 0 und 23
	Minute	mm	zwischen 0 und 59
	Sekunde	ss	zwischen 0 und 59

Die CBM-Echtzeit-Uhr kann alle 24 Stunden des Tages anzeigen, so daß anders als mit 12-Stunden-Uhren eine Unterscheidung zwischen Vormittag und Nachmittag möglich ist. Für hh können daher die Stunden von 00 bis 23 eingegeben werden. Stundenangaben 00 bis 11 bezeichnen den Vormittag (amerikanisch a.m.),und Stundenangaben von 12 bis 23 bezeichnen den Nachmittag (amerikanisch p.m.) mit einer Rückkehr auf 00 um Mitternacht. Beim Übergang von einem 24-Stunden-Zyklus in den anderen, d.h. um Mitternacht, werden hh, mm, ss auf den Wert 0 gesetzt.

Zum Stellen der Echtzeit-Uhr, zum Beispiel nach einem Zeitzeichen, geben Sie die Zeit bereits vor dem Zeitzeichen mit TIME$ ein und lösen beim Zeitzeichen die Uhr durch Drücken der Taste RETURN aus:

```
TIME$ = "130000"
```

LESEN DER ECHTZEIT-UHR

Zum Ablesen der Uhr geben Sie folgenden PRINT-Befehl ein:

```
?TIME$
```

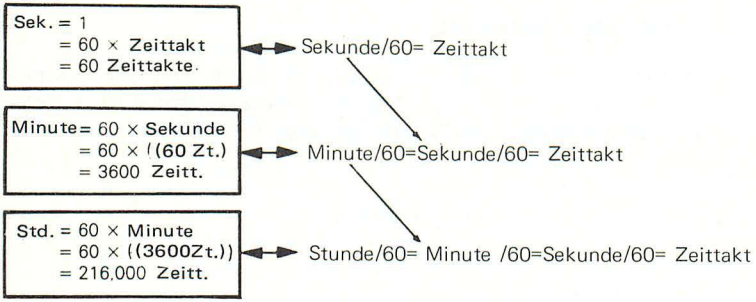
Der Computer stellt die Zeit in Stunden, Minuten und Sekunden auf dem Bildschirm dar:

```
?TIME$
120200
```

Die Echtzeit-Uhr hört auf zu arbeiten, wenn der CBM-Computer ausgeschaltet wird. Beim Wiedereinschalten des Computers muß die Uhr daher neu gesetzt werden.

ARBEITEN MIT DEM ZEITTAKT-GEBER

Der CBM-Computer bildet in Wirklichkeit Zeitangaben aus Zeittakten. Jeder Zeittakt entspricht 1/60 einer Sekunde. TIME, oder TI ist eine reservierte numerische Variable, deren Wert mit jedem Zeittakt automatisch erhöht wird. TIME wird bei Inbetriebnahme des Computers auf Null gesetzt und springt nach 51,839,999 Zeittakten auf Null zurück. TIME$ ist eine Zeichenkette, die aus TIME gebildet wird. Wenn TIME$ aufgerufen wird, dann stellt der Computer die Zeit zwar in Stunden, Minuten und Sekunden (hh, mm, ss) dar, tatsächlich aber wandelt er die gezählten Zeittakte in diese Zeitangabe um. Beachten Sie noch einmal: TIME$ und TI$ sind nicht bloß die Darstellung von TIME oder TI als Zeichenketten, sie wandeln vielmehr die bei der Zeittaktzählung entstehende Zahl TIME (oder TI) in die Form einer Tageszeit-Angabe um. Diese Umwandlung geschieht wie folgt. Jede Sekunde wird in 60 Zeittakte geteilt. Die Minute besteht aus 60 Sekunden. Eine Stunde umfaßt 60 Minuten. Daher entspricht eine Sekunde 60 Zeittakten, eine Minute entspricht 3.600 Zeittakten und eine Stunde entspricht 216,000 Zeittakten:



Die folgenden Befehle wandeln die Ablesung des Zeittaktzählers J in die Tageszeit-Angabe Stunden H, Minuten M und Sekunden S um. Die Befehlsfolge wird begleitet von einer ausführlichen Beschreibung:

```
10 J=TI                              Berechnung der Stunden H
20 H=INT(J/216000)

30 IF H<>0 THEN J=J-H*21600          Berechnung der Minuten M
                                     J=verbleibende Zeittakte
                                     nach Abzug der Stunden
40 M=INT(J/3600)

50 IF M<>0 THEN J=J-M*3600           Berechnung der Sekunden S
                                     J=verbleibende Zeittakte nach Abzug der Minuten
60 S=INT(J/60)
```

```
8 PRINT"■DARSTELLUNG DER UHRZEIT":PRINT:PRINT
10 J=TI
15 T$=TIME$
20 H=INT(J/21600)
30 IF H<>0 THEN J=J-H*21600
40 M=INT(J/3600)
50 IF M<>0 THEN J=J-M*3600
60 S=INT(J/60)
```

```
70 H$=RIGHT$(STR$(H),2)
80 M$=RIGHT$(STR$(M),2)
90 S$=RIGHT$(STR$(S),2)
100 PRINT"H:M:S:  ";H$;":";M$;":";S$,"TIME$:  ";T$
110 PRINT"■■■■";:GOTO10
```

In den Zeilen 70 bis 90 des obigen Programms werden die numerischen Antworten H, M und S in eine für die Darstellung auf dem Bildschirm geeignete Zeichenkette umgewandelt. Zeile 100 veranlaßt sowohl die Ausgabe der im Programm berechneten als auch der über TIME$ ablesbaren Zeit. Die auf dem Bildschirm erscheinenden Zeitangaben sind identisch.

Um eine Vorstellung von der Geschwindigkeit der Zeittakte und ihrer Umwandlung in eine Zeitangabe zu erhalten, können Sie folgendes Programm eingeben. Es stellt die laufende Zeit in Form von TIME$ und TIME (TI) dar:

```
5 REM **VERGLEICH ZWEIER ZEITANGABEN**
10 PRINT"■UHRZEIT :  ":PRINT:PRINT"ZEITTAKT:  "
20 FOR I=1 TO 235959
30 PRINT "■";TAB(13);TIME$
40 FOR J=1 TO 60 STEP 2
50 PRINT"■■■■";TAB(12);TI
60 NEXTJ
70 NEXT I
```

Zwischen zwei Zeitangaben von TIME$ vergeht eine Sekunde. In dieser Zeit könnten 60 Ablesungen des Zeittakt-Zählers TI ausgegeben werden. Die hierfür vorgesehene FOR-NEXT-Schleife mit der Laufvariablen J liest jedoch aus folgenden Gründen nur jeden zweiten Zeittakt (STEP 2 in Zeile 40) ab:

1. Die Darstellung von 60 Ablesungen je Sekunde wäre zu schnell für das Auge.
2. Die Darstellung jeder Ablesung auf dem Bildschirm dauert länger als ein Zeittakt. Hierdurch wird die Rückkehr in die äußere FOR-NEXT-Schleife verzögert und die Anzeigen TIME$ erfolgen langsamer als es sein sollte. Durch STEP 2 werden von den 60 möglichen Zeittakten nur 30 abgelesen, wodurch sich das Problem mit den Verzögerungen verringert. Lassen Sie das Programm laufen, und Sie werden sehen, daß die Zeittakt-Ablesung sich innerhalb einer Sekunde um 60 erhöht. Ohne STEP 2 in Zeile 40 können Sie die Zeitverzögerung in der Darstellung TIME$ beobachten:

Zeitangabe		
	als Uhrzeit	006704
	in Zeittakten	25500

Die Zeitdarstellung durch Zeittakte ist sehr nützlich beim Stoppen von Programmlaufzeiten. Damit können Sie die Leistungsfähigkeit eines Programms feststellen. Betrachten Sie folgendes kurze Programm:

```
10 PRINT"■** TESTPROGRAMM **":PRINT
20 FOR I=32 TO 127
30 PRINT CHR$(I);
40 NEXT I
50 FOR J=161 TO 255
60 PRINT CHR$(J);
70 NEXT J
80 PRINT:PRINT:PRINT"** TETSTENDE **"
```

Die Laufzeit dieses Programms können wir wie folgt bestimmen:

1. Die Zeit TI wird am Programmanfang abgelesen und in einer Variablen A gespeichert.

2. Die Zeit T wird am Programmende abgelesen und in einer Variablen B gespeichert.
3. Die Differenz B - A drückt die Programmlaufzeit als Zahl der erfolgten Zeittakte aus. Die folgende Programmauflistung zeigt die drei erforderlichen Einfügungen:

```
      10 PRINT"□** TESTPROGRAMM **":PRINT
1.) → 15 A=TI
      20 FOR I=32 TO 127
      30 PRINT CHR$(I);
      40 NEXT I
      50 FOR J=161 TO 255
      60 PRINT CHR$(J);
      70 NEXT J
2.) → 75 B=TI
      80 PRINT:PRINT:PRINT"** TETSTENDE **"
3.) → 100 PRINT:PRINT"TI = ";B-A
```

In Zeile 15 wird die Zeit TI abgelesen und ihr Wert in A gespeichert:

```
15 A=TI
```

A = TI [b001762]

A [b001762]

Dann erfolgt der Programmlauf, währenddessen sich TI um 60 Einheiten je Sekunde erhöht. In Zeile 75 erfolgt erneut die Ablesung von TI und die Speicherung des Werts in B:

```
75 B=TI
```

B=TI [b001953]

B= [b001953]

In Zeile 100 erfolgt die Differenzbildung der Zeitablesungen:

B [b001953]

− A [b001762]

191

Im gezeigten Beispiel hat es 151 Zeittakte gedauert, um alle Symbole des Tastenfeldes auf dem Bildschirm darzustellen. Wird die Taktzeit 191 dividiert durch 60 (die Zahl der Zeittakte je Sekunde), dann ergibt sich eine Programmlaufzeit von 3.1833 Sekunden. Hier das Ergebnis eines Programmlaufs:

```
** TESTPROGRAMM **
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]↑← !"#$%&'()*+,-./
0123456789:;<=>?

 ** TETSTENDE **

 TI =  197
```

VERGRÖSSERTE BILDSCHIRMANZEIGE DER UHRZEIT

Das folgende Programm wurde zum Spaß geschrieben.
Es variiert die Zeitanzeige im CBM-Computer durch Verwendung vergrößerter Zahlen 0-9, die mit Hilfe graphischer Symbole gebildet werden. Wegen der Größe des Bildschirms kann das Programm nur die Stunden und Minuten ausgeben. Das Programm ist zwar lang, wie Sie sehen können, aber es besteht fast ausschließlich aus PRINT-Befehlen zur Darstellung der Zahlen. Beobachten Sie nach Eingabe des Programms seine Zeitanzeigen:

```
100 PRINT"
110 S=INT(TIME/60)
120 M=INT(S/60)
130 H=INT(M/60)
140 M=M-H*60
150 T=H
160 GOSUB500
170 PRINT"
180 T=M
190 GOSUB500
200 PRINT"
210 GOTO110
500 U=T-10*INT(T/10)
510 T=INT(T/10)
520 D=T+1
530 GOSUB600
540 D=U+1
550 GOSUB600
560 RETURN
600 ON D GOSUB 1000,1100,1200,1300,1400,1500,1600,
    1700,1800,1900
610 RETURN
1000 PRINT"
1001 PRINT"
1002 PRINT"
1003 PRINT"
1004 PRINT"
1005 PRINT"
1006 PRINT"
1007 PRINT"
1008 PRINT"
1009 PRINT"
1010 RETURN
1100 PRINT"
1101 PRINT"
1102 PRINT"
1103 PRINT"
1104 PRINT"
1105 PRINT"
1106 PRINT"
1107 PRINT"
1108 PRINT"
1109 PRINT"
1110 RETURN
1200 PRINT"
1201 PRINT"
1202 PRINT"
```

```
1203 PRINT"■  ■   ■  ■■■■■■■■■■@";
1204 PRINT"    ■  ■■■■■■■■■■@";
1205 PRINT"   ■  ■  ■■■■■■■■■■@";
1206 PRINT"  ■  ■  ■■■■■■■■■■@";
1207 PRINT" ■  ■   ■■■■■■■■■■@";
1208 PRINT"■       ■■■■■■■■■■@";
1209 PRINT"■        ■  ]]]]]]]]]]";
1210 RETURN
1300 PRINT" ■     ■■■■■■■■■■@";
1301 PRINT"■      ■■■■■■■■■■@";
1302 PRINT"■  ■   ■■  ■■■■■■■■■■@";
1303 PRINT"     ■  ■■■■■■■■■■@";
1304 PRINT"   ■    ■■■■■■■■■■@";
1305 PRINT"   ■    ■■■■■■■■■■@";
1306 PRINT"     ■■  ■■■■■■■■■■@";
1307 PRINT"■  ■■   ■  ■■■■■■■■■■@";
1308 PRINT"■■      ■■■■■■■■■■@";
1309 PRINT" ■■    ■■   ]]]]]]]]]]";
1310 RETURN
1400 PRINT"    ■  ■  ■■■■■■■■■■@";
1401 PRINT"   ■  ■  ■■■■■■■■■■@";
1402 PRINT"  ■  ■  ■■■■■■■■■■@";
1403 PRINT" ■    ■  ■■■■■■■■■■@";
1404 PRINT"■  ■■■  ■  ■■■■■■■■■■@";
1405 PRINT"■  ■■  ■  ■■■■■■■■■■@";
1406 PRINT"■       ■■■■■■■■■■@";
1407 PRINT"■       ■■■■■■■■■■@";
1408 PRINT"    ■  ■  ■■■■■■■■■■@";
1409 PRINT"    ■  ■   ]]]]]]]]]]";
1410 RETURN
1500 PRINT"■       ■■■■■■■■■■@";
1501 PRINT"■       ■■■■■■■■■■@";
1502 PRINT"■  ■     ■■■■■■■■■■@";
1503 PRINT"■      ■  ■■■■■■■■■■@";
1504 PRINT"■       ■■■■■■■■■■@";
1505 PRINT"     ■■  ■■■■■■■■■■@";
1506 PRINT"      ■  ■■■■■■■■■■@";
1507 PRINT"■  ■■   ■  ■■■■■■■■■■@";
1508 PRINT"■■      ■■■■■■■■■■@";
1509 PRINT" ■■    ■■   ]]]]]]]]]]";
1510 RETURN
1600 PRINT" ■     ■■■■■■■■■■@";
1601 PRINT"■      ■■■■■■■■■■@";
1602 PRINT"■  ■   ■■  ■■■■■■■■■■@";
1603 PRINT"■  ■     ■■■■■■■■■■@";
1604 PRINT"■      ■  ■■■■■■■■■■@";
1605 PRINT"■       ■■■■■■■■■■@";
1606 PRINT"■  ■   ■■  ■■■■■■■■■■@";
1607 PRINT"■  ■■   ■  ■■■■■■■■■■@";
1608 PRINT"■■      ■■■■■■■■■■@";
1609 PRINT" ■■    ■■   ]]]]]]]]]]";
1610 RETURN
1700 PRINT"■       ■■■■■■■■■■@";
1701 PRINT"■       ■■■■■■■■■■@";
1702 PRINT"     ■  ■■■■■■■■■■@";
1703 PRINT"    ■  ■■■■■■■■■■@";
1704 PRINT"   ■  ■  ■■■■■■■■■■@";
1705 PRINT"  ■  ■  ■■■■■■■■■■@";
1706 PRINT"  ■  ■   ■■■■■■■■■■@";
1707 PRINT"  ■  ■   ■■■■■■■■■■@";
1708 PRINT"  ■  ■   ■■■■■■■■■■@";
```

```
1709 PRINT"  ▯ ▮       ┌┬┬┬┬┬┬┬┬┐";
1710 RETURN
1800 PRINT" ▰▰      ▰▰ ▮▮▮▮▮▮▮▮▮▮▯";
1801 PRINT"▰▰        ▰▰▮▮▮▮▮▮▮▮▯";
1802 PRINT"▯  ▰▰   ▰▯  ▰▮▮▮▮▮▮▮▮▯";
1803 PRINT"▯  ▰▰   ▰▰  ▰▮▮▮▮▮▮▮▮▯";
1804 PRINT"▰▯        ▰▰▮▮▮▮▮▮▮▮▯";
1805 PRINT"▰▰        ▰▰▮▮▮▮▮▮▮▮▯";
1806 PRINT"▯  ▰▰   ▰▯  ▰▮▮▮▮▮▮▮▮▯";
1807 PRINT"▯  ▰▰   ▰▰  ▰▮▮▮▮▮▮▮▮▯";
1808 PRINT"▰▯        ▰▰▮▮▮▮▮▮▮▮▯";
1809 PRINT" ▰▯     ▰▰   ┌┬┬┬┬┬┬┬┬┐";
1810 RETURN
1900 PRINT" ▰▰      ▰▰ ▮▮▮▮▮▮▮▮▮▯";
1901 PRINT"▰▰        ▰▰▮▮▮▮▮▮▮▮▯";
1902 PRINT"▯  ▰▰   ▰▯  ▰▮▮▮▮▮▮▮▮▯";
1903 PRINT"▯  ▰▰   ▰▰  ▰▮▮▮▮▮▮▮▮▯";
1904 PRINT"▰▯        ▰▮▮▮▮▮▮▮▮▯";
1905 PRINT" ▰▯       ▰▮▮▮▮▮▮▮▮▯";
1906 PRINT"        ▯  ▰▮▮▮▮▮▮▮▮▯";
1907 PRINT"▯  ▰▰   ▰▰  ▰▮▮▮▮▮▮▮▮▯";
1908 PRINT"▰▯        ▰▰▮▮▮▮▮▮▮▮▯";
1909 PRINT" ▰▯     ▰▰   ┌┬┬┬┬┬┬┬┬┐";
1910 RETURN
```

ZUFALLSZAHLEN

In CBM-Computern wird ein Algorithmus verwendet, der ausgehend von einer Startzahl eine Folge von Zufallszahlen erzeugt. Eine unveränderte Startzahl führt immer zur gleichen Folge von Zufallszahlen.

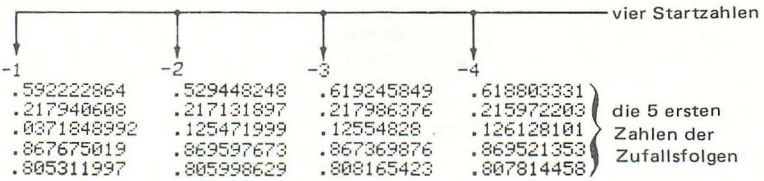
STARTZAHL EINER ZUFALLSFOLGE

Jeder CBM-Computer erzeugt bei Inbetriebnahme eine gleichbleibende Startzahl. Diese Startzahl unterscheidet sich möglicherweise von einem CBM-Computer zum nächsten. Jeder einzelne CBM-Computer erzeugt daher nach Inbetriebnahme eine unveränderliche, von seiner Startzahl ausgehende Folge von Zufallszahlen. Die folgende Bildschirmwiedergabe zeigt die ersten fünf Zahlen einer typischen Zufallsfolge, wie Sie nach Inbetriebnahme mit der Zufalls-Funktion RND(arg) gebildet wird:

```
### COMMODORE BASIC ###

 7167 BYTES FREE

READY.

FOR I=1 TO 5:? RND(1):NEXT
RUN
 .880969862
 .355265655
 .659512252
 .803285178
 .546991144
READY.
```

Außer dieser Anfangsfolge von Zufallszahlen, auf die Sie keinen Einfluß haben, kann der CBM-Computer unzählig viele weitere Zufallsfolgen erzeugen, die Sie durch Eingabe eines negativen Arguments in der Funktion RND(arg) bestimmen kön-

nen. Da es unzählig viele negative Zahlen gibt, gibt es auch unzählig viele Listen von Zufallszahlen, die ein CBM-Computer erzeugen kann. Diese Zufallsfolgen sind zwar von einem CBM-Computer zum nächsten verschieden, für jeden einzelnen CBM-Computer bleiben sie jedoch unveränderlich:



Die Erzeugung einer Zufallsfolge kann mit jeder BASIC-Anweisung erfolgen, die die Funktion RND mit negativem Argument enthält. Sie können einen einfachen Zuweisungs-Befehl benutzen wie:

```
20 X=RND(-2)
```

Wird die Funktion RND mit negativem Argument in einer BASIC-Anweisung angesprochen, dann erfolgt die Rücksetzung eines Zeigers auf die erste Zufallszahl derjenigen Liste, die dem negativen Argument zugeordnet ist. Zum Beispiel: Auf unserem CBM-Computer bewirkt die oben genannte Zuweisung (Zeile 20) immer die Ausgabe der Zahl.592222864 als erste Zufallszahl der Liste mit der Startzahl –2.

Auch bei wiederholtem Ansprechen der Funktion RND wird also bei unverändertem negativen Argument immer die gleiche Zufallsfolge gebildet.

Dem gleichen negativen Argument entsprechen jedoch in verschiedenen CBM-Computern verschiedene Zufallsfolgen.

AUSLESEN DER ZUFALLSZAHLEN

Die Auswahl einer Zufallsfolge erfolgt über ein negatives Argument in der Funktion RND; zum Auslesen der Zufallszahlen einer gewählten Folge wird die gleiche Funktion RND benutzt, jedoch mit einem positiven Argument. In beiden Fällen wird RND über eine BASIC-Anweisung angesprochen. Ein Beispiel: Das folgende Programm wählt fünf Zufallsfolgen mit den Startzahlen –1 bis –5 aus und liest die jeweils sechs ersten Zufallszahlen jeder Folge aus:

```
30 FOR I=-1 TO -5 STEP -1
35 X=RND(I):PRINTI
40 FOR J=1 TO 5
50 PRINT RND(1)
60 NEXT J
70 NEXT I
100 STOP
```

Die äußere FOR-NEXT-Schleife dient der Auswahl einer Zufallsfolge; die BASIC-Anweisung X = RND(I) in Zeile 35 bewirkt das Rücksetzen eines Auslesezeigers auf die erste Zufallszahl der Folge. Die innere FOR-NEXT-Schleife (Zeilen 40, 50 und 60) liest die ersten sechs Zufallszahlen einer Folge aus:

```
-1
 .592222864
 .217940608
 .0371848992
 .867675019
 .805311997
-2
 .529448248
 .217131897
```

```
 .125471999
 .869597673
 .805998629

-3
 .619245849
 .217986376
 .12554828
 .867369876
 .808165423
-4
 .618803331
 .215972203
 .126128101
 .869521353
 .807814458
-5
 .529738186
 .216918235
 .128416908
 .868422708
 .717787951
```

Um Ihnen zu zeigen, daß jede Zufallsfolge im CBM-Computer aus einer unveränderlichen Liste von Zufallszahlen besteht, unterbrechen Sie im eben besprochenen Programm die Erzeugung und das Auslesen von Zufallszahlen und starten es erneut. Hierzu folgende Programmänderung:

```
30 FOR I=-1 TO -5 STEP -1
35 X=RND(I):PRINTI
40 FOR J=1 TO 3
50 PRINT RND(1)
60 NEXT J
70 FOR K=10 TO 11
75 PRINT RND(1)
80 NEXT K
90 NEXT I
100 STOP
```

Dieses Programm gibt wieder die ersten sechs Zufallszahlen von fünf Zufallsfolgen aus, aber die Ausgabe erfolgt in zwei getrennten FOR-NEXT-Schleifen (Zeilen 40 und 70).

Dennoch stellt dieses Programm die gleichen Zufallszahlen wie das zuvor besprochene Programm dar. In anderen Worten: jedesmal, wenn in der Befehlsfolge eines Programms die Funktion RND mit einem positiven Argument auftritt, wird die nächste Zufallszahl aus derjenigen Zufallsfolge ausgelesen, die durch die letzte im Programm genannte RND-Funktion mit negativem Argument ausgewählt wurde; jedesmal, wenn in der Befehlsfolge eines Programms eine RND-Funktion mit negativem Argument angetroffen wird, springt der Auslesezeiger auf die erste Zufallsrate der im Argument von RND bezeichneten Zufallsfolge. Ein Beispiel: fügen Sie im zuletzt besprochenen Programm die Zeile 65 mit folgender Anweisung ein:

```
65 X=RND(I)
```

Wenn Sie jetzt das Programm laufen lassen, werden die ersten drei Zufallszahlen in jeder Zufallsfolge ausgelesen (FOR-NEXT-Schleife mit Laufindex J), danach werden die ersten drei Zufallszahlen noch einmal ausgelesen (FOR-NEXT-Schleife mit Laufindex K).

Probieren Sie diese Programme aus, indem Sie es in Ihren Computer eingeben. Verändern Sie sowohl die negativen Argumente (Startzahlen) über den Laufindex I in der äußeren FOR-NEXT-Schleife, als auch die Anzahl der auf dem Bildschirm dargestellten Zufallszahlen über die Laufindices J und K in den inneren FOR-NEXT-Schleifen. Führen Sie diese Versuche fort, wenn Sie zufrieden sind mit Ihrem Verständnis für die Art, in der Zufallszahlen erzeugt werden.

AUSDRUCKEN VON ZUFALLSZAHLEN

Zum einfacheren Vergleich sollten Sie besser Zufallszahlen ausdrucken, als auf dem Bildschirm darstellen (vorausgesetzt, Sie haben einen Drucker). Obwohl die Programmierung von Druckerausgaben erst in Kapitel 6 beschrieben wird, haben wir die erforderlichen Anweisungen zum Ansprechen eines Druckes schon hier zusammen gestellt:

```
10 OPEN 4,4
20 CMD 4
30 FOR I=-1 TO -5 STEP -1
35 X=RND(I):PRINTI
40 FOR J=1 TO 5
50 PRINT RND(1)
60 NEXT J
70 NEXT I
80 PRINT#4
90 CLOSE 4
100 STOP
```

Die Anweisungen in Zeilen 10 und 20 stellen die Verbindung zum Drucker her; die Anweisungen in Zeilen 80 und 90 lösen diese Verbindung wieder auf. Stellen Sie sicher, daß der Drucker in Betrieb ist und richtig mit Ihrem CBM-Computer verbunden wurde. Machen Sie dann Versuche mit dem Ausdrucken von Zufallszahlen, indem Sie das eben genannte Programm abwandeln. Die Ausdrucke zu jedem Versuch machen es für Sie einfacher, die Zufallszahlen verschiedener Programmvariationen zu vergleichen.

Wenn Sie die RND-Funktion mit dem Argument 0 verwenden, wird die Erzeugung der Zufallszahlen von der in CBM-Computern eingebauten Uhr abhängig gemacht. Es gibt jedoch Ähnlichkeiten zwischen den Zufallszahlen von RND-Funktionen mit dem Argument 0. Hiervon können Sie sich überzeugen, wenn Sie in Zeile 50 des oben stehenden Programms das Argument der RND-Funktion von 1 in 0 ändern und das Programm einige Male ablaufen lassen. Sie werden zwar feststellen, daß keine zwei Zufallsfolgen identisch sind, aber doch sehr ähnlich.

ERZEUGEN VON ZUFÄLLIGEN STARTZAHLEN

Damit die gleichen Zufallszahlen sich nicht bei jedem Programmlauf widerholen, brauchen Sie eine Programmöglichkeit, auch Startzahlen zufällig zu verändern. Eine Möglichkeit besteht in der Verwendung des augenblicklichen Zeittaktes, TI:

```
10 X=RND(-TI): REM STARTZAHL ZIEHEN
```

Bei jeder erneuten Ausführung der Anweisung in Zeile 10 wird eine andere Zufallsfolge ausgewählt.

Eine nahezu völlig zufällige Startzahl erhalten Sie mit einem Argument der Form RND(–RND(00)). Die Funktion RND(0) ist jedoch nur auf CBM-Computern mit BASIC 2.0 verfügbar:

```
10 X=RND(-RND(0))
```

Auch hier wird bei jeder erneuten Ausführung der Anweisung in Zeile 10 eine unterschiedliche Zufallsfolge gestartet.

Das Programm im folgenden Abschnitt benutzt als Argument –TI, das für alle Versionen von CBM BASIC gültig ist. Arbeitet Ihr Computer mit CBM BASIC 2.0, dann können Sie auch das Argument –RND(0) statt –TI verwenden.

ZUFALLSZAHLEN AUSSERHALB DES BEREICHS NULL BIS EINS

CBM-Computer erzeugen Zufallszahlen im Bereich von 0 bis 1. Hieraus können Sie leicht Zufallszahlen machen, die einen anderen Bereich überdecken. Ein Beispiel: Nehmen Sie an, die Zufallszahlen wollen die Werte 1, 2, . . . 6 annehmen, wie in einem Würfelspiel. Das erste ist die Multiplikation der Zufallszahlen des CBM-Computers mit 6:

```
6*RND(1)
```

Hierdurch entstehen Zufallszahlen in Fließkomma-Darstellung, die größer als 0 und kleiner als 6 sind ($0 < n < 6$). Das zweite ist die Addition einer 1, so daß Zufallszahlen im Bereich $1 < n < 7$ entstehen:

```
,*RND(1)+1
```

Der Bereich der Zufallszahlen stimmt jetzt mit dem Bereich der Würfelzahlen überein. Die Zufallszahlen müssen lediglich noch in Ganzzahlen umgewandelt werden, so daß sie den Bereich von 1 bis 6 in Gestalt von Ganzzahlen überdecken:

```
INT(6*RND(1)+1)
```

oder:

```
A%=6*RND(1)+1
```

Die Funktion RND liefert Bruchzahlen, für deren Umwandlung in Ganzzahlen folgende allgemein gültige Befehle zu verwenden sind. Beachten Sie, daß die Funktion INT nur Zahlen im Bereich ± 32767 verarbeiten kann:

INT(n+1)*RND(1))	Bereich 0 bis n
INT(n*RND(1)+1)	Bereich 1 bis n
INT(n–m+1)*RND(1)+m	Bereich m bis n

Machen Sie jetzt Versuche mit Zufallszahlen, die in unterschiedlichen Intervallen liegen, indem Sie die eben angegebenen Anweisungen abwandeln.

Im jetzt folgenden Programm wird der Zeittakt – TI zur Bildung zufälliger Startzahlen benutzt. Dieses Programm bildet Zufallszahlen im Intervall M bis N. Die Zahlenwerte M und N werden in Zeile 10 vor einem Programmlauf eingegeben. Beachten Sie, daß diese Werte auch negativ sein können. Im folgenden Beispiel wird auf dem Bildschirm eine unendliche Folge von Zufallszahlen aus dem Intervall – 50 bis + 50 dargestellt. (Zum Abbrechen des Programms drücken Sie die Taste STOP). Die Zufallsfolge wird bei jedem Programmlauf verschieden sein, da – TI zufällige Startzahlen bildet.

```
10 M=-50:N=50
20 X=RND(-TI):PRINT X
30 FOR I=1 TO 8
40 C%=(N-M+1)*RND(1)+M
50 PRINT C%;:NEXT I
60 PRINT:GOTO 30
RUN
 8.27633085E-06
-14   9 -34 -35 -47 -44   28   31
 29  -8 -36 -28 -42 -28   15   14
  7 -13   3  -8   8   41   19  -43
 35   12   24  -7  -7 -21  -47    1
-32 -49   7 -49   28 -22  -17  -24
-12   7   27   1   11   9  -18   35
 48   49   1   34 -46 -29  -43   29
-18   5  -30   2   8  -28  -13  -23
 48 -15  -12 -45   26   44 -25   2
-9   4   27   50   33 -16 -43  -15
 20   20   17   43 -18 -48 -38   24
-16   43 -50   36 -38   5   11   25
-30   6  -25 -47   32   10   42  -21
```

```
-47 -38 -28 -8  16 -20  42 -4
-34  36 -17  27 -8 -49 -6 -35
-19  19 -35  48 -42  36 -25  2
-49  37  47  38 -20 -25  32 -50
-5 -35 -35  17 -41  36 -19  4
 33 -20  45 -7  48 -4 -33 -10
 1  27 -39 -14 -38 -6  4  10
-5  17  2  49  0 -40 -5  32
-50  32 -24 -37 -38  22 -13 -27
-24 -30  35  10  6  16 -50  49
-49  50  43  38 -21  47 -43  28
 32 -35 -18 -5  27 -46 -14  23
-49 -45  27  7 -35  1  46 -25
-8  20 -8 -12 -46 -31 -17 -18
-47  47 -49  18  47  17  40 -13
-40  48 -41 -33  5 -14 -46  45
-29 -37  22  17  42  33 -31  49
 8 -4  36  37  11  18  29  25
 0 -1  2 -16  32 -29 -31  33
-9 -41 -4  47  12 -22  9 -48
-40  32  15  32 -50  3 -9  19
```

Ändern Sie die Werte von M und N in Zeile 10 des eben beschriebenen Programms, um Zufallszahlen in unterschiedlichen Intervallen sichtbar zu machen. Zum Beispiel: mit M = 1, N = 6 wird eine unendliche Folge von Zufallszahlen im Intervall 1 bis 6 erzeugt. Derartige Zufallsfolgen treten beispielsweise beim Würfeln auf.

ZUFÄLLIGE VERTAUSCHUNGEN (MISCHEN VON SPIELKARTEN)

Ein kurzes Überfliegen der im vorigen Abschnitt wiedergegebenen Zufallszahlen zeigt, daß von den 101 möglichen Zahlen im Intervall – 50 bis + 50 manche Zahlen wiederholt vorkommen, andere überhaupt nicht. Derartige Zahlenfolgen geben beispielsweise sehr gut die Ergebnisse eines Würfelspiels wieder. Ein anderer Anwendungsfall für Zufallszahlen besteht darin, alle Zahlen aus einem gegebenen Intervall in zufälliger Reihenfolge und ohne Wiederholung wiederzugeben. Nehmen Sie als Beispiel das Mischen eines Stapels von Spielkarten.

Das folgende Programm zeigt eine Möglichkeit, das Mischen von Spielkarten auf dem CBM-Computer durchzuführen. Dieses Programm erzeugt eine zufällige Vertauschung der Zahlen 1 bis 52, die dann in Tabellenform auf dem Bildschirm dargestellt werden. Die Zufallszahlen können auf beliebige Weise den Spielkarten zugeordnet werden, etwa so:

A=1, 2=2, 3=3, . . ., Q=12, K=13
Pik = 0, Herz = 13, Karo = 26, Kreuz = 42

Nach diesem Schema entspricht Pik-As = 1 + 0 = 1, As-König 12 + 0 = 12, Herz-Bube 3 + 13 = 16 usw.

In Tabelle FL des Mischprogramms wird über die ausgegebenen Zufallszahlen Buch geführt, um wiederholte Ausgaben einer Zufallszahl zu vermeiden. Beachten Sie, daß genau 52 Zahlen ausgegeben werden, und sich keine der Zahlen wiederholt. Jeder Programmlauf erzeugt eine neue Zufallsfolge der Zahlen 1 bis 52:

```
10 DIM FL(52),D%(52)
20 X=RND(-TI):PRINT X
30 FOR I=1 TO 52
40 C%=52*RND(1)+1
50 IF FL(C%)<>0 GOTO 40
60 D%(I)=C%:FL(C%)=1
70 PRINT C%;
80 NEXT I
RUN
```

```
 1.18586613E-05
 48  40  13  37  50  43  46  31  49  44
 23  38  25  11  9  35  32  30  24  41
26  5  6  1  45  10  21  14  42  20  15
 34  18  52  47  7  16  8  19  33  36  4
  17  3  22  27  29  28  39  2  51  12
RUN
 1.01154728E-06
 14  35  52  50  26  48  27  36  34  25
 18  20  41  33  39  7  46  24  23  28  1
9  3  12  43  2  31  44  4  1  32  37  3
0  40  22  45  48  42  49  16  11  6  10
  29  9  51  17  8  15  38  5  21  13
```

Die Ausgabegeschwindigkeit dieses Programms wird langsamer, je mehr Zahlen bereits dargestellt sind. Die Ausgabe der letzten Zahl dauert besonders lang. Das hängt damit zusammen, daß das Programm mehr und mehr Zufallszahlen erzeugen muß, bis es eine gefunden hat, die nicht schon ausgegeben wurde. Eine einfache Befehlsfolge wie diese ist natürlich noch offen für Verbesserungen. Der Programmlauf kann beschleunigt werden, indem die letzte Zahl nicht durch Ziehen von Zufallszahlen sondern durch einen Blick auf die bereits ausgegebene Tabelle gefunden wurde.

ZUFÄLLIGE BILDSCHIRMDARSTELLUNGEN

Im folgenden verwenden wir das früher besprochene (Kapitel 2) Programm BLANKET. Das Programm wird so abgewandelt, daß es den Bildschirm nicht mehr zeilenweise mit einem über das Tastenfeld eingegebenen Symbol füllt, sondern durch zufällige Verteilung der Symboldarstellungen über die insgesamt 1000 Positionen des Bildschirms.

Hier die erste Version dieses Programms:

```
10 REM ******  B L A N K E T  **********
20 REM * ZUFALLSVERTEILTE DARSTELLUNG *
30 REM * EINES AM TASTENFELD EINGE-   *
40 REM * GEBENEN SYMBOLS               *
50 REM ***********************************
90 PRINT"BITTE SYMBOL EINGEBEN ODER MIT RETURN BEENDEN:";
100 GET C$:IF C$="" GOTO 100
105 IF  C$=CHR$(13) GOTO 170
110 PRINT"□";     :REM BILDSCHIRM LOESCHEN
120 X=RND(-TI)    :REM NEUE STARTZAHL ZIEHEN
125 C=(ASC(C$)AND128)/2 OR (ASC(C$)AND63)
127 A=1000*RND(1)+32768
130 POKE A,C      :REM SYMBOL DARSTELLEN
140 GET D$:IF D$="" GOTO 127
150 C$=D$
160 GOTO 105
170 END
```

Bis Zeile 110 wird wie im Programm BLANKET ein neues Symbol eingelesen und der Bildschirm gelöscht. Die Anweisung in Zeile 120 speichert eine neue Startzahl für die nachfolgende Zufallsverteilung der Bildschirmdarstellungen. Jede Bildschirmdarstellung wird ausgeführt mit der Funktion POKE, die ein darzustellendes Symbol C in zufällig wechselnde Adressen A des Bildschirmspeichers einliest. Die

Anweisung in Zeile 125 wandelt das als Zeichenkette eingegebene C$ in den entsprechenden Kode C der Funktion POKE um, wobei mit ASC zunächst eine Umwandlung in den ASCII-Code erfolgt und dann mit den logischen Operatoren AND, OR der in Tabelle A-4, Anhang A wiedergegebene Zusammenhang zu den Kode-Nummern der Funktion POKE hergestellt wird. Die Anweisung in Zeile 127 bestimmt eine zufällige Adresse im Bildschirmspeicher in Form einer Zahl im Bereich 32768 bis 33767; hierzu wird das Intervall 0 bis 1 der von RND gebildeten Zufallszahlen mit m = 32768 und n = 33767 wie folgt geändert:

```
A =  (n-m+1)*RND(1)+m                    Berechnung der zu-
     (33767-32768+1)*RND(1)+32768        fälligen Bildschirm-
  =    1000*RND(1)+32768                 adresse in Zeile 127
```

Die Adresse A wird als Bruchzahl ausgegeben, die weder mit der Funktion INT noch durch Typumwandlung mit A% = A in eine Ganzzahl verwandelt werden kann, da die Bildschirmadressen genau hinter der größten darstellbaren Ganzzahl 32767 beginnen. Die Funktion POKE hat zum Glück die Eigenschaft, von jeder Bruchzahl nur ihren ganzzahligen Anteil zu verwenden. In anderen Anwendungsfällen der Zufallsfunktion RND haben sie jedoch die Beschränkung ganzzahliger Zufallszahlen auf den Bereich – 32767 bis + 32767 zu beachten.

Die erste Version des oben wiedergegebenen Programms füllt den Bildschirm mit einem eingegebenen Symbol durch eine Folge zufällig streuender Bildschirmausgaben. Hierzu bedient es sich der Funktion POKE, mit der ein Symbol in zufällig wechselnde Adressen des Bildschirmspeichers geschrieben wird. Es kommt daher vor, daß ein Symbol mehrfach in die gleiche Bildschirmadresse geschrieben wird, während andere Adressen leer bleiben. Das Programm setzt seine Symbolausgaben fort, selbst wenn der Bildschirm bereits gefüllt ist, bis ein neues Symbol über das Tastenfeld eingegeben wird.

Wenn Sie das Programm laufen lassen, beobachten Sie, daß sehr schnell etwa die Hälfte der Bildschirmpositionen mit dem Symbol beschrieben wird. Dann verlangsamt sich die Wiedergabe neuer Symbole, bis schließlich die letzten Symbole des fast völlig gefüllten Bildschirms äußerst langsam eingetragen werden. Es kann 3 Minuten dauern, bis diese Programmversion den Bildschirm völlig gefüllt hat.

Die Arbeitsgeschwindigkeit des Programms ändert sich nicht, aber gegen Ende des Programmlaufs verliert das Programm an Wirkung, da die meisten der Bildschirmpositionen bereits beschrieben sind. Die wiederholte Wiedergabe eines Symbols an der gleichen Bildschirmposition führt zu keiner sichtbaren Wirkung.

Die Programmlaufzeit kann beträchtlich verkürzt werden, wenn die überflüssige Symbolausgabe an bereits beschriebene Bildschirmpositionen vermieden wird. Das wird mit einer zweiten Version des Programms BLANKET erreicht. Die zweite Version des Programms erzeugt Zufallszahlen nicht mehr im gesamten Intervall von 1000 Bildschirmplätzen, sondern nur in Intervallen noch freier Bildschirmplätze. Hierzu registriert das Programm die Adressen noch freier Bildschirmplätze in einer Tabelle T(A%), auf die es über zufällig gezogene Tabellenadressen A% zugreift. Die Zahl verfügbarer Tabellenadressen A% verringert sich mit jeder neuen Symbolausgabe um 1. Hier das Programm:

```
10 REM ******  B L A N K E T  **********
20 REM * ZUFALLSVERTEILTE DARSTELLUNG *
30 REM * EINES AM TASTENFELD EINGE-    *
40 REM * GEBENEN SYMBOLS  -VERSION 2- *
50 REM ************************************
70 DIM T(999)
80 GOSUB 200       :REM INITIIEREN DER ZUFALLSFOLGE
90 PRINT"BITTE SYMBOL EINGEBEN ODER MIT RETURN BEENDEN:";
100 GET C$:IF C$="" GOTO 100
```

```
105 IF  C$=CHR$(13) GOTO 170
110 PRINT"□";     :REM BILDSCHIRM LOESCHEN
120 X=RND(-TI)    :REM NEUE STARTZAHL ZIEHEN
125 C=(ASC(C$)AND128)/2 OR (ASC(C$)AND63)
126 FOR N=999 TO 0 STEP -1
127 A%=(N+1)*RND(1):REM ZUFAELLIGE ADRESSE A% BILDEN
128 A=T(A%)+32768  :REM ADRESSE IM BILDSCHIRMSPEICHER
129 TP=T(A%):T(A%)=T(N):T(N)=TP
130 POKE A,C          :REM SYMBOL DARSTELLEN
140 NEXT N
150 REM
160 GOTO 100
170 END
199 REM UNTERPROGRAMM ZUR INITIIERUNG DER ZUFALLSFOLGE
200 FOR I=0 TO 999:T(I)=I:NEXT
210 RETURN
```

In Zeile 70 erfolgt die Dimensionsangabe der Tabelle T(A%), die für 1000 Adressen entsprechend 1000 Bildschirmplätzen ausgelegt ist.

In Zeile 80 erfolgt der Sprung in ein Unterprogramm, das die Zahlen 0 bis 999 in die entsprechenden Plätze der Tabelle T einliest, wobei T(0) die Zahl 0 enthält, T(1) die Zahl 1, . . . T(999) die Zahl 999. Da der Zugriff auf die Tabelle über eine zufällig gezogene Adresse A% erfolgt, ist das Einlesen der Zahlen in aufsteigender Ordnung ohne Bedeutung für das Anlegen der Tabelle und ergibt sich nur durch Verwenden der Einleseschleife FOR-TO.

Die Zeilen 90 bis 125 entsprechen genau der ersten Version des Programms.

In Zeilen 126 bis 140 dient eine FOR-NEXT-Schleife zum Füllen von 1000 Bildschirmplätzen mit dem am Tastenfeld eingegebenen Symbol. Zeile 126 enthält den Lauf-Index N, der sich nach jeder Symbolausgabe um den Wert 1 verringert, womit sich auch das Intervall für die Erzeugung von Zufallszahlen verringert. In Zeile 127 wird eine zufällige Tabellenadresse A% im Bereich 0 bis N der noch freien Bildschirmplätze gebildet. In Zeile 128 wird die absolute Adresse A im Bildschirmspeicher berechnet, die sich aus der Anfangsadresse 32768 des Bildschirmspeichers und dem unter T(A%) gefundenen Zahlenwert in der Tabelle T zusammensetzt.

In Zeile 129 wird die Tabelle freier Bildschirmplätze bereinigt. Der unter T(A%) vorgefundene Bildschirmplatz wird gegen den noch freien, unter T(N) am Tabellenende stehenden Bildschirmplatz vertauscht. Hierdurch wird gleichzeitig die Tabelle freier Bildschirmplätze um eine Eintragung kleiner, was sich auch in der Verringerung des Laufindex M um den Wert 1 ausdrückt, ausgelöst durch die Anweisung NEXT N in Zeile 140.

In Zeile 130 erfolgt schließlich die Ausgabe des am Tastenfeld eingegebenen Symbols C an der zufälligen Bildschirmposition A.

KAPITEL 6

Peripherie–Geräte zu CBM Computern: Datasette Bandgerät, Floppy Disk Speicher und Drucker

Ein Computer-System besteht aus mehr als dem Tastenfeld, dem Bildschirm und dem Computer selber. Um ein Programm nicht jedesmal neu eingeben zu müssen, wenn Sie es laufen lassen wollen, können Sie es auf einem Floppy Disk-Speicher oder einer Bandkassette speichern. Sie können dann das Programm in den Speicher des Computers laden, um es dort ablaufen zu lassen, wodurch Sie eine wiederholte Programmeingabe vermeiden (siehe Kapitel 2).

Genauso können Sie auch Daten speichern. Denken Sie an ein Programm für Versandadressen. Dieses Programm würde man auf eine Bandkassette oder einer Diskette abspeichern. Ein Programm für Versandadressen wird benutzt, um eine Liste von Namen und Adressen ausgeben zu können. Diese Namen und Adressen sind wie das Programm ebenfalls auf einer Bandkassette oder Diskette gespeichert. Schließlich werden Namen und Adressen wieder von der Bandkassette oder der Diskette gelesen, um Adressen-Etiketten zu drucken. Genau hierfür wird der Drucker benötigt.

Die meisten Computer-Systeme verfügen über einen Zeilendrucker. Zeilendrucker werden benutzt, um Programmergebnisse auszudrucken, wie zum Beispiel die Adressen auf den Etiketten. Ein Zeilendrucker ist aber auch für die Programmerstellung unentbehrlich. Die wirkungsvollste Art der Änderung und Korrektur eines Programms besteht im Ausdrucken des Programms in seiner augenblicklichen Form, um beabsichtigte Änderungen in die gedruckte Programmauflistung einzutragen. Danach werden die niedergeschriebenen Programmänderungen über das Tastenfeld eingegeben.

In diesem Kapitel beschäftigen wir uns mit der Beschreibung der Programmlogik, die erforderlich ist, um Datasette-Bandgeräte, Floppy Disk-Speicher und Drucker in CBM-BASIC anzusprechen.

CBM-Computer besitzen einen Steckeranschluß für den IEEE 488 Bus, d.h. ein mehradriges Verbindungskabel, das für den Datenaustausch mit dem Computer von mehreren Peripheriegeräten gemeinschaftlich benutzt werden kann. Dieser von der amerikanischen Industrie standardisierte Bus wird zum Anschluß von Floppy Disk-Speichern und Druckern benutzt. Der IEEE 488 Bus wird in industriellen Anwendungen auch zum Anschluß von Meßinstrumenten und elektronischen Sondergeräten benutzt. Wir werden zwar Floppy Disk-Speicher und Drucker beschreiben, nicht aber den IEEE 488 Bus selbst, noch Programmiertechniken, mit denen Meßgeräte über diesen Bus angesprochen werden können.

DATENSPEICHERUNG AUF MAGNETISCHEN TRÄGERN

DAS KONZEPT EINER DATEI

Information wird in Form von "Dateien" auf Bandkassetten oder Disketten gespeichert.

Um die Idee einer "Datei" zu verstehen, denken Sie an ein Bücherregal. Die Bandkassette oder Diskette ist zu vergleichen mit dem Regal; jedes Buch im Regal entspricht einer Datei.

Für den Benutzer eines Computers ist das Konzept einer Datei sehr einfach. Wenn Sie eine Datei "öffnen", wird Ihnen alle in der Datei gespeicherte Information zugänglich. Die Information bleibt zugänglich, bis sie die Datei "schließen". Das entspricht dem Vorgang, ein Buch aus dem Regal zu nehmen und zu öffnen. Aber anders als beim Buch ist das Beschreiben einer Datei genauso einfach wie das Lesen einer Datei. Wenn der Computer ein Programm oder Daten auf eine Bandkassette oder Diskette schreibt, wird jeweils eine neue Datei geschaffen, oder an eine alte angefügt.

Eine Datei kann jeden Umfang besitzen, der lediglich durch Speicherkapazität der Bandkassette oder Diskette beschränkt ist. Sie können eine neue Datei einrichten und nichts eintragen, so daß in diesem Falle die Datei leer ist. Das entspricht dem Fall, daß ein Buch nur aus Buchdeckeln besteht, aber keinen Inhalt hat. Eine Datei muß auf eine einzelne Bandkassette oder eine Diskette passen. Daher hängt die größte Dateilänge nur von der Speicherkapazität der Bandkassette oder der Diskette ab.

Jede Bandkassette oder Diskette kann bis zu 256 Dateien enthalten. Wenn so viele Dateien auf einer einzelnen Bandkassette oder Diskette gespeichert sind, ist natürlich die Länge der einzelnen Datei sehr kurz.

Der verfügbare Speicherplatz in Ihrem CBM-Computer hat keinen Einfluß auf die Größe einer Datei. Eine Datei kann sehr viel länger sein als der Speicherumfang Ihres Computers. Nach "Öffnung" einer Datei können Sie ein einzelnes Symbol von ihr lesen, oder soviel Information, wie in dem verfügbaren Speicherplatz Ihres Computers Platz ist . Beim Beschreiben einer Datei wird häufig Information, die aus dem Speicher des Computers ausgelesen wird, zu Informationen hinzugefügt, die bereits auf der Bandkassette oder der Diskette gespeichert waren.

PROGRAMM-DATEIEN

Es gibt zwei Arten von Dateien: Programm-Dateien und Dateien für Daten. Eine Programm-Datei enthält Programm-Anweisungen, wie der Name schon sagt.

Eine Programm-Datei wird jedesmal neu eingerichtet, wenn Sie ein Programm mit dem BASIC-Befehl SAVE auf einer Bandkassette oder Diskette aufbewahren sollen. Eine Programm-Datei wird gelesen, wenn ein Programm mit dem BASIC-Befehl LOAD in den Speicher des Computers geladen wird. Diese Operationen wurden in Kapitel 2 beschrieben.

Jede Datei kann einen Namen tragen; der Name einer Programm-Datei wird zugleich auch Name des Programms. CBM-Computer lassen Datei-Namen zu, die aus bis zu 128 Buchstaben bestehen, wovon die ersten 16 Buchstaben wiedergegeben werden. Datei-Namen auf Disketten dürfen aus höchstens 16 Buchstaben bestehen. Es ist daher ein guter Vorsatz, sämtliche Datei-Namen auf 16 Buchstaben zu beschränken.

Der verfügbare Speicherplatz in Ihrem CBM-Computer bestimmt die größte Länge einer Programm-Datei. Wenn Sie mit SAVE ein Programm auf Bandkassette

oder Diskette speichern, wird eine einzige Programm-Datei erzeugt, deren Inhalt später beim Laden des Programms in den Speicher des Computers zurückgelesen wird. Sie können nicht lediglich einen Teil der Programm-Datei in den Arbeitsspeicher Ihres Computers laden. Daher muß die größte Länge der Programm-Datei kleiner sein als die Speicherkapazität Ihres CBM-Computers. Wenn Sie ein sehr langes Programm haben, das nicht in den verfügbaren Speicherplatz des Computers paßt, können Sie es in eine Zahl von Dateien zerlegen, von der jede im Computer Platz hat. Am Ende des Programmlaufs jedes Programmabschnitts laden Sie einfach den nächsten Abschnitt in den Arbeitsspeicher des Computers und lassen ihn ablaufen. Auf diese Weise erreichen Sie den Ablauf des gesamten Programms. Im Verlauf dieses Kapitels werden wir die Schritte beschreiben, um sehr lange Programme in der genannten Weise ablaufen zu lassen.

Ein Vorteil von Programm-Dateien ist, daß Sie nichts über ihren inneren Aufbau zu wissen brauchen. Wenn Sie ein Programm auf Bandkassette oder Diskette speichern, nimmt es die Gestalt einer Datei an, die Sie später wieder in den Arbeitsspeicher des Computers zurückholen können. Hierzu müssen Sie lediglich in der Lage sein, die Programm-Datei zu identifizieren, etwa über den Datei-Namen oder über die Kenntnis der Stelle, an der die Datei steht.

DATEN-DATEIEN

Eine Datei für Daten enthält, wie der Name vermuten läßt, Informationen, die als Daten interpretiert werden, im Gegensatz zu Programmanweisungen. Daten-Dateien werden erzeugt, beschrieben und gelesen durch Programme.

DATEN-AUFZEICHNUNGEN UND DATEN-FELDER

Dateien für Daten werden unterteilt in "Aufzeichnungen", die wiederum unterteilt sind in "Felder".

Ein einzelnes Feld enthält soviel Informationen, wie durch einen einzelnen Variablennamen dargestellt werden kann. Daher kann ein einzelnes Feld eine Ganzzahl, eine Gleitzahl, oder eine Zeichenkette enthalten.

Eine Aufzeichnung besteht aus einem oder mehreren Feldern. Aufzeichnungen stellen häufig sich wiederholende Informationseinheiten innerhalb einer Datei dar, aber das muß nicht notwendigerweise so sein.

Betrachten Sie eine Liste mit Versandadressen. Die gesamte Liste von Versandadressen bildet eine Datei. Jede Eintragung von Name und Adresse in die Datei stellt eine Aufzeichnung dar. Wenn Name und Adresse mit dem in Kapitel 5 beschriebenen Programm eingegeben werden, dann besteht jede Aufzeichnung aus 4 Feldern: Name, Straße, Stadt und Land. Diese Datei-Organisation wird in Figur 6-1 gezeigt.

Eine Datei kann aus einer oder mehreren Aufzeichnungen bestehen. Jede Aufzeichnung kann sich aus einem oder mehreren Feldern zusammensetzen. Die Zahl der Aufzeichnungen in einer Datei und die maximale Länge einer Aufzeichnung schwankt mit dem Datei-Typ, wie später in diesem Kapitel beschrieben wird. In allen Fällen ist jedoch die Dateilänge nur durch die Speicherkapazität der Diskette begrenzt.

Für die Länge einer Aufzeichnung auf Bandkassette bestehen keine Beschränkungen. Eine Aufzeichnung kann jede Länge haben, die auf die Bandkassette paßt.

INFORMATIONSAUSTAUSCH ZWISCHEN BANDKASSETTE/DISKETTE UND COMPUTER

Anfangs wird man beim Arbeiten mit Dateien auf Bandkassetten oder Disketten das Gefühl haben, daß etwas falsch läuft. Man erwartet, daß die Bandkassette in Antwort auf jeden Lese- oder Schreibbefehl eine Bewegung ausführt. Manchmal beobachten Sie derartige Bewegungen, zu anderen Zeiten stehen die Geräte still. Die Erklärung ist, daß der Datenaustausch zwischen Computer und Bandkassette bzw. Diskette über einen Pufferspeicher erfolgt.

Wenn der Computer Daten von einem dieser Geräte liest, werden jeweils soviele Daten gelesen, wie der Pufferspeicher faßt. Sie sehen dann für einige Zeit keine Bewegung der Speichergeräte, bis das Programm den Pufferspeicher geleert hat und wieder auf die Speichergeräte zugreifen muß.

Beim Schreiben in eine Bandkassette oder Diskette werden die Daten zuerst in den Pufferspeicher übertragen. Sobald der Pufferspeicher voll ist, wird der gesamte Speicherinhalt des Puffers an die Bandkassette oder Diskette ausgegeben: und bei dieser Gelegenheit sind die Speichergeräte in Betrieb. Die Bewegungen setzen dann wieder solange aus, bis der Pufferspeicher erneut gefüllt ist.

Der Pufferspeicher für das Bandgerät befindet sich im Arbeitsspeicher des CBM-Computers. Er ist 192 Byte lang und kann 191 Byte Daten speichern. Der Pufferspeicher für Disketten befindet sich im Floppy Disk-Speichergerät und nicht im Arbeitsspeicher des CBM-Computers. Der Pufferspeicher eines Floppy Disk-Speichers kann 256 Symbole speichern. Die Pufferspeicher für Bandkassette und Diskette sind relativ groß. Folglich sind die Antriebe dieser Geräte für die meiste Zeit nicht in Betrieb, während der der Computer den Pufferspeicher ausliest oder neu beschreibt.

DATEINAMEN UND GERÄTENUMMERN

Wir benutzen den Ausdruck "Ein/Ausgabe-Programmierung" zur Beschreibung einer Programmlogik, die den Datenaustausch zwischen Computer und Peripheriegeräten bewerkstelligt. Bandgeräte, Floppy Disk-Speicher und Drucker sind Beispiele für Peripheriegeräte oder externe Geräte.

Um eine Ein/Ausgabe-Operation auszuführen, muß die Programmlogik das externe Gerät identifizieren können, mit dem ein Datenaustausch erfolgen soll. Das mag Ihnen nicht überraschend vorkommen. Wie sieht es aber für den Computer aus? Denken Sie an dieses Problem in Begriffen der Programmierung: Programme können Daten oder Variable oder Konstanten in einer BASIC-Anweisung identifizieren. Die Programmlogik zum Datenaustausch mit dem Computer wird daher in ähnlichen Begriffen formuliert. Das Konzept ist am einfachsten zu verstehen, wenn Sie sich auch das Tastenfeld und den Bildschirm eines CBM-Computers als externe Geräte vorstellen, was sie in der Tat auch sind. Die Ausführung eines INPUT-Befehls besteht darin, vom Tastenfeld kommende Daten Variablen zuzuordnen, deren Namen im INPUT-Befehl aufgeführt sind. Ein Beispiel: bei Ausführung der Anweisung

```
10 INPUT A
```

wird eine am Tastenfeld eingegebene Zahl der Gleitkomma-Variablen A zugeordnet. Ganz ähnlich die PRINT-Anweisung, mit der Variable oder Konstanten an den Bildschirm ausgegeben werden können. Ein Beispiel: bei Ausführung der Anweisung

```
20 PRINT A
```

wird die der Gleitkomma-Variablen A zugewiesene Zahl auf dem Bildschirm dargestellt.

Die Anweisungen INPUT und PRINT benutzen für den Datenaustausch des Computers einen Variablen-Namen (in diesem Falle der Gleitkommavariablen A). Bei Ausführung einer INPUT-Anweisung nimmt der Computer als externes Gerät das Tastenfeld an. Bei Ausführung einer PRINT-Anweisung nimmt der Computer als externes Gerät den Bildschirm an.

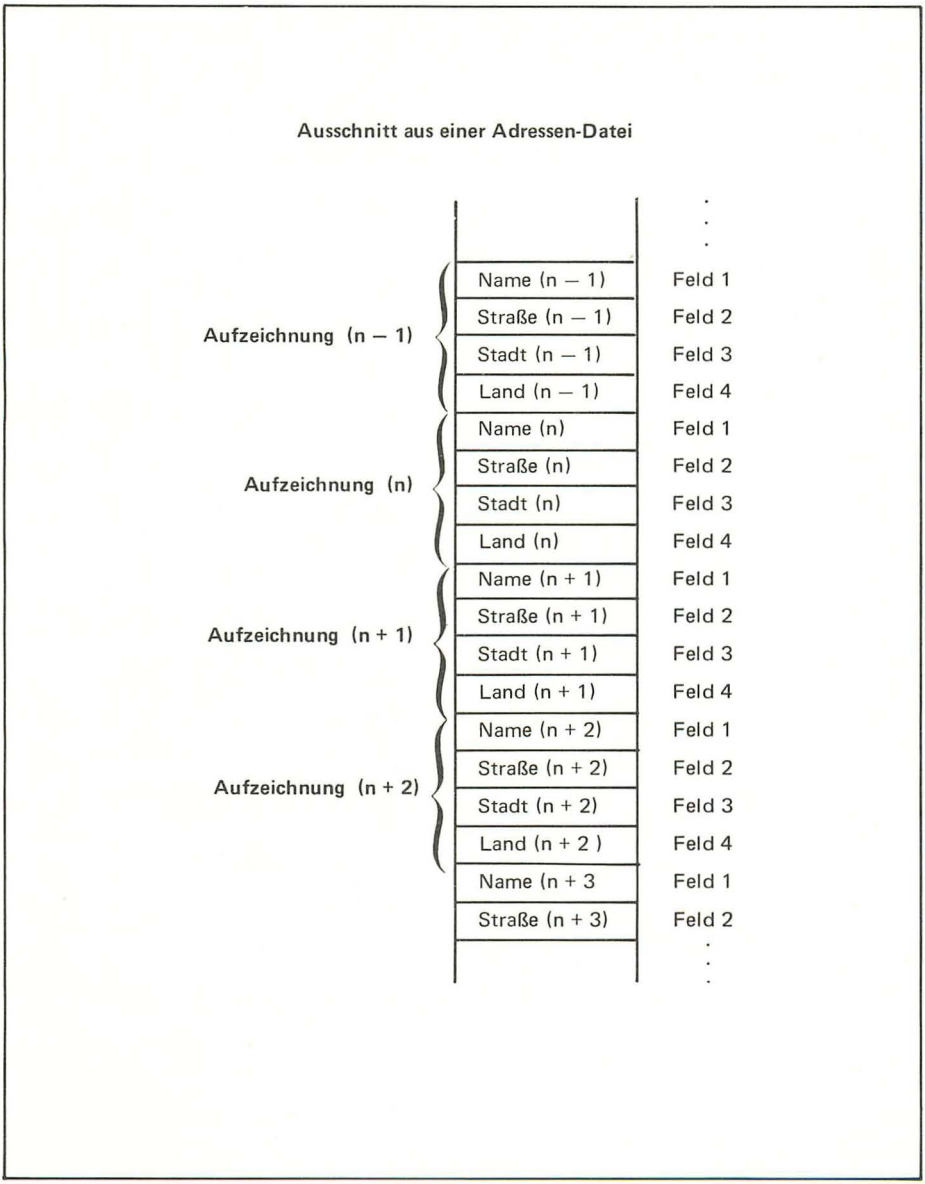


Bild 6-1. Veranschaulichung der Aufzeichnungen in einer Datei

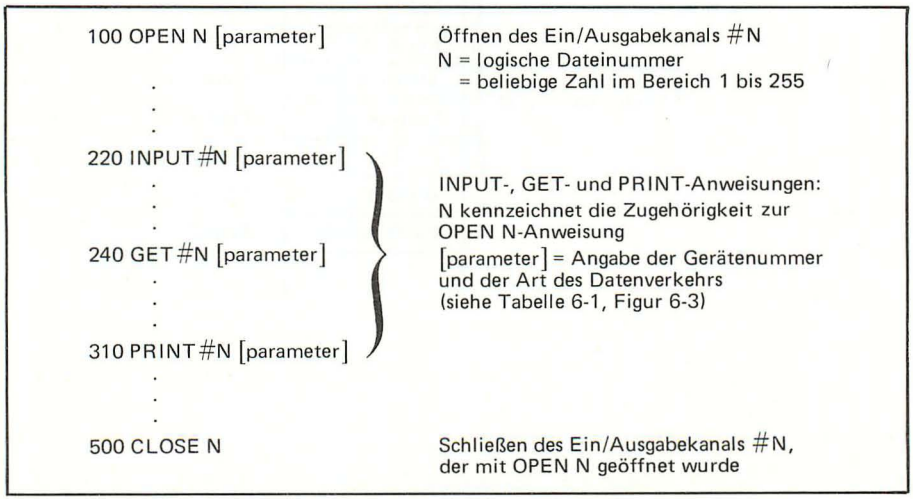
Ein/Ausgabe-Programmierung wird verwickelter, wenn es sich um den Datenaustausch mit Bandgeräten, Floppy Disk-Speichern, Druckern und anderen externen Geräten außer dem Tastenfeld und dem Bildschirm handelt. Für diese verwickelteren Ein/Ausgabe-Operationen müssen Sie zwischen dem Programm und dem gewählten externen Gerät zunächst einen "Kanal" für den Datenaustausch öffnen. Nach Abschluß der erforderlichen Ein/Ausgabe-Operationen müssen Sie diesen Kanal dann wieder schließen. In CBM-BASIC werden einzelne Kanäle durch Kanalnummern identifiziert, die die Werte 0 - 255 annehmen können.

Sie öffnen einen Kanal in CBM-BASIC mit der Anweisung OPEN; Parameter in der OPEN-Anweisung bezeichnen das anzusprechende externe Gerät und die Art des Datenaustausches, wie Figur 6-2 darstellt. Bis zum Schließen des Kanals ist für jede Eingabe oder Ausgabe lediglich die Angabe der Kanalnummer erforderlich, um den Datenaustausch eindeutig zu beschreiben.

Jedes externe Gerät hat seine eigene unverwechselbare Gerätenummer. Beim Öffnen eines Kanals wird diese Gerätenummer verwendet. Kanalnummern bezeichnen dagegen keine festen Zuordnungen. Kanalnummern werden daher auch oft "logische Datei-Nummern" oder "logische Geräte-Nummern" genannt.

Der Name "logische Datei" gibt das Wesen eines Kanals sehr genau wieder, da ein Kanal eine Verbindung herstellt zwischen Programm und Datei, die beide nichtphysikalischer, sondern logischer Natur sind.

Logische Dateinummern stellen eine Programmierhilfe dar. Wie Figur 6-2 zeigt, können Sie mit der Anweisung OPEN eine Eingabe- oder Ausgabe-Operation vorschreiben. Einer der Parameter in der Anweisung OPEN ist die Kanalnummer oder logische Dateinummer; die weiteren Parameter in der OPEN-Anweisung bezeichnen das externe Gerät, die anzusprechende Datei und die Art des Datenaustausches. Nach Abschluß der Ein- oder Ausgabe-Operation schließen Sie den Kanal mit einer CLOSE-Anweisung. Die CLOSE-Anweisung benötigt nur einen Parameter: die Kanalnummer oder die logische Dateinummer. Diese logische Dateinummer bezieht die CLOSE-Anweisung auf die zugehörige OPEN-Anweisung. Alle Ein/Ausgabe-Anweisungen zwischen der OPEN- und CLOSE-Anweisung benutzen lediglich die Kanalnummer oder logische Dateinummer, um das angesprochene externe Gerät und die Art des Datenaustausches zu beschreiben.



Figur 6-2 Die Funktion der "logischen Dateinummer N" in einem BASIC-Programm

Die logische Dateinummer verbindet die Anweisungen OPEN, CLOSE, INPUT, GET und PRINT innerhalb eines Programms.

Sobald Sie eine logische Dateinummer in einer OPEN-Anweisung benutzt haben, können Sie die gleiche logische Dateinummer nicht mehr für die Eröffnung eines anderen Kanals benutzen, ehe nicht die logische Datei geschlossen wurde. Tun Sie es dennoch, dann meldet sich CBM-BASIC mit einem Syntax-Fehler. Im übrigen bestehen keine Beschränkungen für die Art der Vergabe logischer Dateinummern innerhalb Ihres Programms.

Gerätenummern beschreiben das externe Gerät, das vom Computer angesprochen werden soll. Die Gerätenummer erscheint als Parameter in der OPEN-Anweisung. Jedes externe Gerät, das mit dem CBM-Computer kommunizieren kann, erhält eine unveränderliche Gerätenummer. Sobald der CBM-Computer auf eine Gerätenummer in einer OPEN-Anweisung stößt, spricht er über eine geeignete elektronische Logik das mit der Gerätenummer bezeichnete Gerät an und ermöglicht den Datenaustausch. Tabelle 6-1 gibt eine zusammenfassende Darstellung externer Geräte und zugehöriger Gerätenummern, wie sie vom CMB-Computer benutzt werden. Es sind 256 Geräte-Nummern mit den Werten 0 - 255 verfügbar. Gegenwärtig werden jedoch nur die Gerätenummern 0 - 30 benutzt, wie Tabelle 6-1 zeigt.

Tabelle 6-1. Parameter für Gerätebezeichnung und Art des Datenverkehrs

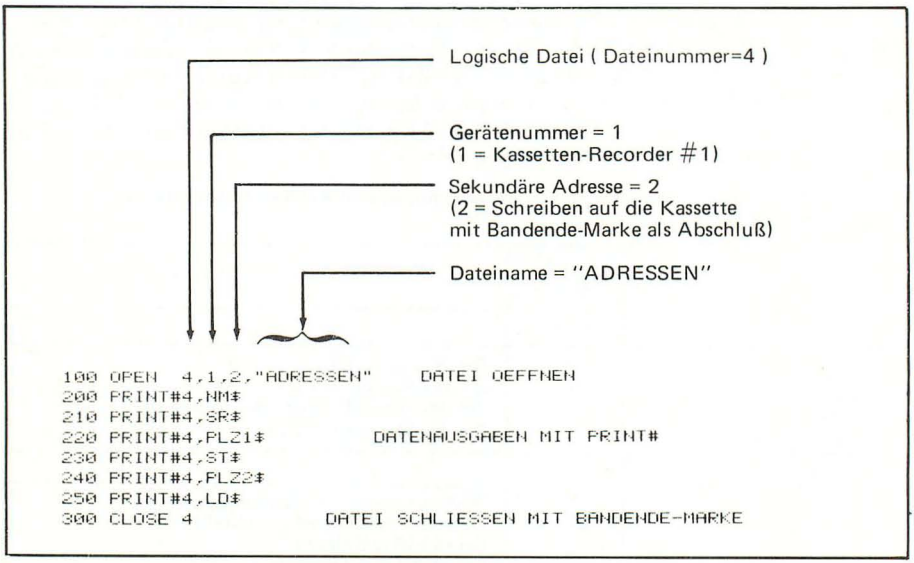
Gerät	Gerätenummer	Sekundäre Adresse	Art des Datenverkehrs
Tastatur	0	keine	
Bandgerät #*	1 +)	0	Öffnen zum Lesen
		1	Öffnen zum Schreiben
Bandgerät #2	2	2	Öffnen zum Schreiben, Abschluß mit Bandende-Marke (EOT)
Bildschirm	3	keine	
Drucker (alle Modelle)	4	0	Daten drucken wie empfangen
		1	Daten im definierten Druckbild drucken
		2	Druckbild definieren
		3	Zeilen je Seite vorgeben
		4	Funktionstest des Druckers ermöglichen
		5	Sonderzeichen bilden
		6	Zeilenabstand vergrößern (nur Modell 2022)
Floppy Disk-Speicher (alle Modelle)	8	0	Laden eines Programms in den Computer
		1	Speichern eines Programms vom Computer
		2-14	ohne Zuweisung
		15	Öffnen (OPEN) eines Kontroll-/Status-Kanals
Weitere Geräte am IEEE 488-Bus	5, 6, 7 und 9 bis 31		Gerätenummern und Sekundäre Adressen für den Betrieb am IEEE 488-Bus abhängig vom Gerätehersteller
	32 bis 255 gegenwärtig nicht verfügbar		

* = gerade angeschlossenes Gerät

+) bei fehlender Parameterangabe

Die meisten externen Geräte reagieren nicht nur auf ihre Gerätenummer, sondern auch auf eine Reihe von "Sekundäradressen". Sekundäre Adressen stellt man sich am besten als "Befehle" vor, mit denen der Computer dem externen Gerät mitteilt, welche Operationen es ausführen soll. Für üblicherweise mit CBM-Computern verbundene externe Geräte wurden die Sekundären Adressen in Tabelle 6-1 zusammengestellt. Sie sollten sich im Augenblick nicht mit Sekundären Adressen beschäftigen; wir werden später die Programmierung von Eingaben und Ausgaben ausführlich beschreiben, wobei durch häufigen Gebrauch die Funktion Sekundärer Adressen sichtbar wird.

Figur 6-3 gibt ein vollständiges Bild des Gebrauchs von Parametern in Eingabe/Ausgabe-Anweisungen.



Figur 6-3 Anwendungsbeispiel für Parameter-Angaben

Die fünf PRINT#-Anweisungen in Zeilen 200 - 240 schreiben die fünf Felder einer Aufzeichnung von Name und Adresse die Datei ADRESSEN auf einer Bandkassette des externen Geräts mit der Gerätenummer 1. Bei jedem PRINT#-Befehl weiß der Computer, was zu tun ist, weil er die logische Dateinummer nach dem Zeichen # prüft. In Figur 6-3 ist diese logische Dateinummer 4; die OPEN-Anweisung mit dieser Nummer steht in Zeile 100. Wenn der Computer keine OPEN-Anweisung mit der benötigten logischen Dateinummer findet, würde er keine Eingabe oder Ausgabe versuchen, da er nicht weiß, was er tun soll. Die OPEN-Anweisung in Figur 6-3 enthält die logische Dateinummer 4 und die Gerätenummer 1, womit Bandgerät 1 angesprochen wird. Die Sekundäre Adresse 2 bestimmt im vorliegenden Fall, daß auf die Bandkassette in Bandgerät 1 geschrieben, nicht aber von der Bandkassette gelesen werden kann. Überdies wird am Ende des Schreibvorgangs eine Bandendemarke auf die Bandkassette geschrieben, um die Eintragung weiterer Daten zu verhindern. Die Anweisung OPEN gibt der zu eröffnenden Datei den Namen MAILLIST.

In Zeile 300 steht eine CLOSE-Anweisung. Diese CLOSE-Anweisung enthält die logische Dateinummer 4, womit alles, was die OPEN-Anweisung in Zeile 100 eröffnet hat, durch die CLOSE-Anweisung in Zeile 300 beendet wird. Da die OPEN-Anweisung in Zeile 100 eine Sekundäre Adressnummer 2 angibt, wird mit Ausführung

der CLOSE-Anweisung in Zeile 300 eine Bandendemarke auf die Bandkassette geschrieben.

Die logische Dateinummer 4 verbindet die Anweisungen in Zeilen 200 bis 300 mit der OPEN-Anweisung in Zeile 100. Zusätzliche Parameter, die in der OPEN-Anweisung in Zeile 100 enthalten sind, beschreiben den Typ der Datenübertragung stellvertretend für alle anderen Anweisungen in Zeilen 200 bis 300.

GERÄTESTATUS

Zeilendrucker können zwar Daten von einem Computer empfangen, sie können aber nicht Daten in den Computer eingeben. Dennoch hindert Sie nichts an der Verwendung einer INPUT-Anweisung mit einer logischen Dateinummer, die schon von einer vorangegangenen OPEN-Anweisung zur Eröffnung der Datenausgabe an einen Drucker benutzt wurde.

Obwohl ein Bandgerät sowohl Daten empfangen als auch an den Computer übermitteln kann, wird über die Sekundäradresse einer OPEN-Anweisung nur eine der Möglichkeiten zugelassen: d.h. entweder Lesen von der Bandkassette oder Schreiben auf die Bandkassette. Sie könnten dennoch fälschlicherweise eine Anweisung geben, mit der Sie das Bandgerät entgegen der vereinbarten Richtung der Datenübertragung ansprechen.

Wenn Sie eine PRINT-, GET- oder INPUT-Anweisung verwenden, mit der von einem externen Gerät eine Operation verlangt wird, zu der dieses Gerät entweder unfähig ist, oder für die es keine Programmanweisung erhalten hat, dann wird das externe Gerät einen fehlerhaften Gerätestatus melden. Auf keinen Fall wird ein externes Gerät versuchen, eine Operation auszuführen, die ihm nicht von einer OPEN-Anweisung erlaubt wurde, selbst wenn es hierzu in der Lage wäre. Beispiel: Wenn Sie mit einer OPEN-Anweisung für ein Bandgerät nur Schreiboperationen vorschreiben, dann werden eine INPUT- oder GET-Anweisung nicht ausgeführt; stattdessen wird ein fehlerhafter Gerätestatus gemeldet, und das ist alles.

Nach jeder Eingabe oder Ausgabe und unabhängig von deren Erfolg oder Mißerfolg melden externe Geräte ihren Gerätestatus an den Computer zurück. Der Gerätestatus besteht aus 8 Bit und kann in der Variablen ST abgefragt werden:

```
10 X=ST
```

Diese Anweisung überträgt den augenblicklichen Gerätestatus, was immer der sein mag, in die Variable X.

Tabelle 6-2 enthält eine Übersicht aller Meldungen, mit denen übliche externe Geräte ihren Gerätestatus an den CBM-Computer zurückmelden. Sie sollten sich an Tabelle 6-2 erinnern, wenn Sie später Programme für den Zugriff zu verschiedenen externen Geräten schreiben.

Verwenden Sie den Gerätestatus ST nicht zur Überprüfung der Operationen von Tastenfeld oder Bildschirm, obwohl Tastenfeld und Bildschirm wie externe Geräte eine Gerätenummer haben.

Standardisierte Statusmeldungen auf dem IEEE 488 Bus sind der Vollständigkeit halber in Tabelle 6-2 erwähnt, aber dieses Buch beschreibt nicht den Betrieb von Geräten über diesen Bus.

Tabelle 6-2. Statusmeldungen externer Geräte, abfragbar in der Variablen ST

	Statusmeldung							
	00000001 **Lesen als 1**	**00000010** **Lesen als 2**	**00000100** **Lesen als 4**	**00001000** **Lesen als 8**	**00010000** **Lesen als 16**	**00100000** **Lesen als 32**	**01000000** **Lesen als 64**	**10000000** **Lesen als −128**
Bandgerät # 1 oder # 22 (READ Modus)	Operation OK	Operation OK	zu kurzer Block: gelesener Datenblock kürzer als erwartet	zu langer Block: gelesener Datenblock länger als erwartet	nicht korrigierbarer Lesefehler	Übertragungsfehler. Ein oder mehrere Bits falsch gelesen	Dateiende-Marke gelesen	Bandende-Marke gelesen
Bandgerät # 1 oder # 2 (VERIFY Modus)					Übertragungsfehler entdeckt			
Floppy Disk-Speicher (alle Modelle)	Gerät zum Datenempfänger nicht verfügbar	Gerät zum Datensender nicht verfügbar					Dateiende-Marke gelesen	Floppy Disk-Speicher nicht angeschlossen
IEEE 488-Bus	Zeit abgelaufen für Datenempfänger	Zeit abgelaufen für Datensender					EOJ Ende der Arbeit	Gerät nicht angeschlossen

ARBEITEN MIT DATEIEN AUF BANDKASSETTE

Wir werden jetzt Programmschritte beschreiben, die zum Arbeiten mit Dateien auf Bandkassetten benötigt werden. Wir werden beschreiben, wie Dateien erstellt, gelesen und unter Programmkontrolle modifiziert werden.

Wir werden von einigen BASIC-Anweisungen zum Arbeiten mit Dateien Gebrauch machen, die wir in diesem Buch noch nicht benutzt haben. Seien Sie aber daran erinnert, daß Kapitel 8 eine vollständige Beschreibung aller CBM-BASIC-Anweisungen enthält. Wenn Sie Schwierigkeiten mit Beschreibungen in diesem Kapitel haben, da Sie einer der verwendeten BASIC-Anweisungen nicht verstehen, dann sollten Sie Kapitel 8 konsultieren und die vollständige Beschreibung der Anweisung, mit der Sie Schwierigkeiten haben, lesen, um dann wieder zu diesem Kapitel zurückzukehren.

Sie können den CBM-Computer anweisen, Daten auf eine Bandkassette zu schreiben oder Daten von einer Bandkassette zu lesen, aber Sie können nicht die physikalische Bewegung des Magnetbandes über ein Programm beeinflussen. Es ist wichtig, daß Sie die Arbeitsweise von Bandgeräten verstehen; sonst könnten Sie Operationen versuchen, die das Bandgerät nicht ausführen kann.

Dateien werden sequentiell, d.h. der Reihe nach, auf einer Bandkassette gespeichert. Eine Kopfmarke läuft der ersten Datei voran und eine Bandende-Marke schließt die letzte Datei ab. Jede Datei endet mit einer Dateiende-Marke.

Die Kopfmarke wird automatisch an den Anfang einer Bandkassette geschrieben, falls Sie das erste Mal die Bandkassette beschreiben. Bei dieser Gelegenheit werden Sie Bewegungen der Bandkassette feststellen, die Sie nicht erwartet haben; im übrigen aber ist das Vorhandensein einer Kopfmarke ohne Bedeutung für Sie.

Der Computer kann Dateien auffinden, während sich das Band mit Wiedergabe-Geschwindigkeit (PLAY) bewegt, nicht jedoch bei schnellem Vorlauf (FAST-FORWARD). Das Lesen einer Dateiende-Marke wird mit dem Statuswort ST = 64 zurückgemeldet und das Lesen einer Bandende-Marke mit dem Statuswort −128 (siehe Tabelle 6-2).

Ein Magnetband kann nicht durch den Computer zurückgespult werden, noch kann während des Zurückspulens irgendetwas von der Bandkassette gelesen werden.

Sie haben die Bewegung der Bandkassette durch Drücken von Tasten am Bandgerät selbst zu starten, sobald der CBM-Computer Sie hierzu auffordert. Drücken Sie keine Bedientasten am Bandgerät, bevor Sie durch eine Mitteilung auf dem Bildschirm hierzu angewiesen wurden. Danach hält der Computer automatisch das Bandgerät im richtigen Augenblick an und wiederholt automatisch einen Bandlauf für weitere Zugriffe auf Dateien, falls Sie die Stellung der Bedientasten nicht verändert haben.

Zum Einlesen von Daten in ein Bandgerät muß zuvor die Kassette richtig eingelegt worden sein. Dies liegt in der Verantwortung des Benutzers eines CBM-Computers. Alte Daten auf der Bandkassette werden von neuen Daten überschrieben. Befindet sich zufällig der durchsichtige Bandvorlauf unter dem Schreibkopf, dann kann der Einlesevorgang zwar beginnen, aber es wird nichts gespeichert. Das sicherste Verfahren ist, auf eine Leerkassette zu schreiben und das Band auf den Anfang seiner magnetischen Oberfläche zu positionieren; danach können Sie eine Aufzeichnung nach der anderen einlesen, bis das Ende der Bandkassette erreicht ist. Das Bandgerät selber stellt sicher, daß zwischen dem Ende einer Aufzeichnung und dem Beginn der nächsten genügend Abstand vorhanden ist. Sie brauchen nicht (und sollten auch nicht) das Magnetband nach jeder Aufzeichnung ein Stück vorwärts bewegen, um Abstand zur nächsten Aufzeichnung zu schaffen. Sie können die Bandkassette auch nicht zu-

rücklaufen lassen, um eine Aufzeichnung zu wiederholen, da die Wahrscheinlichkeit sehr gering ist, das Band bis zur richtigen Stelle zurücklaufen zu lassen. Selbst ein kleiner Fehler führt zu Aufzeichnungen durch das Bandgerät, die Sie anschließend nicht mehr lesen können.

Zum Lesen von Dateiaufzeichnungen müssen Sie sicherstellen, daß das Band zu einer Stelle zurückgespult wurde, die vor der ersten zu lesenden Datei liegt. Der CBM-Computer kann zwar jede mit einem Namen versehene Datei finden, während das Band mit Wiedergabegeschwindigkeit abläuft, aber er kann nicht automatisch das Band zurückspulen, um eine zurückliegende Bandeintragung zu finden.

Versuchen Sie niemals, die Aufzeichnung auch nur eines kleinen Teils einer Datei zu wiederholen; dieser Versuch ist einfach zu risikovoll. Ein Beispiel: Nehmen Sie an, Sie haben 10 Namen und Adressen auf einer Bandkassette gespeichert und wünschen jetzt, den fünften Namen mitsamt Adresse zu ändern. Theoretisch könnten Sie die ersten vier Namen und Adressen lesen, wonach die Bandkassette am Anfang des fünften Namen stehenbleiben würde. Dann könnten Sie den fünften Namen samt Adresse überschreiben. In der Praxis ist so etwas selten möglich. Die Bandbewegungen lassen sich nicht sehr genau steuern, und es ist daher sehr wahrscheinlich, daß die Eintragung des neuen Namens etwas zu früh oder etwas zu spät beginnen. Die Folge ist, daß ein kleines Stück der alten Namen- und Adresseneintragung am Anfang oder Ende der neuen Eintragung stehen bleibt und Sie in beiden Fällen nicht in der Lage sind, die neue Eintragung zu lesen.

Um Dateien auf Bandkassette zu ändern, müssen Sie zwei Bandgeräte verwenden. Lesen Sie von der Kassette auf dem einen Bandgerät die alten Daten aus und schreiben Sie die neuen, geänderten Daten in die Kassette auf dem anderen Bandgerät. Sie sollten dieses Verfahren selbst dann verwenden, wenn Sie nur eine einzelne Dateneintragung unter 100 zu ändern haben.

Es gibt keine Anweisungen in CBM-BASIC, mit denen ein Magnetband lediglich bewegt oder positioniert werden kann.

Drei Programmschritte sind für den Zugriff auf Dateien auf Bandkassette erforderlich:

1. Öffnen der Datei (OPEN)
2. Lesen von der Datei (INPUT) oder Schreiben in die Datei (PRINT)
3. Schließen der Datei (CLOSE)

ÖFFNEN EINER DATEI AUF BANDKASETTE : OPEN

Zum Öffnen einer Datei müssen Sie die Anweisung OPEN gebrauchen. Sie erhalten eine Fehlermeldung (SYNTAX-Fehler), wenn Sie versuchen, auf eine ungeöffnete Datei zuzugreifen. Wenn Sie eine Datei auf Bandkassette öffnen, können Sie die OPEN-Anweisung in einem der folgenden Formate verwenden:

OPEN N	Öffnen der logischen Datei N. Lesen der ersten Banddatei auf Bandgerät #1.
OPEN N,D	Öffnen der logischen Datei N. Lesen der ersten Banddatei auf Bandgerät #D. D = 1 für Bandgerät #1, D = 2 für Bandgerät #2.
OPEN N,D,S	Öffnen der logischen Datei N. Zugriff auf erste Banddatei des Bandgeräts #D für eine Operation, die mit der Sekundären Adresse S definiert ist (s. Tabelle 6-1).
OPEN N,D,S, Dateiname	Öffnen der logischen Datei N. Zugriff auf die angegebene Datei in Bandgerät #D für eine durch S (Tabelle 6-1) definierte Operation.

Die OPEN-Anweisung kann mit verschiedenen Parameter-Kombinationen verwendet werden. N ist der einzige Parameter, der vorhanden sein muß Bei fehlender Eingabe wird angenommen, daß D den Wert 1 und S den Wert 0 haben (Ersatzwerte). Fehlt der Datei-Name, dann wird auf die erste Datei auf der Bandkassette zugegriffen.

Wenn die OPEN-Anweisung ausgeführt wird, um von einer Banddatei zu lesen, dann stellt der CBM-Computer folgende Mitteilung auf dem Bildschirm dar, falls am Bandgerät noch keine der Bedientasten betätigt worden ist:

```
PRESS PLAY ON TAPE #1
OK
```

Taste wurde gedrückt
das Magnetband beginnt sich zu bewegen

Nach Bedienung des Bandgeräts liest der CBM-Computer die Bandanfangs-Marke der Bandkassette. In direktem Dialog mit dem Computer, d.h. wenn die OPEN-Anweisung nicht durch ein Programm erfolgte, erscheinen folgende weitere Mitteilungen auf dem Bildschirm (Angaben in Klammern für den Fall, daß mit der OPEN-Anweisung auch ein Dateiname angegeben wurde):

SEARCHING [FOR Dateiname]	
FOUND Dateiname a	Funktion: Suchen nach einer Datei. Hierbei werden
FOUND Dateiname b	die ersten 16 Symbole aller Dateinamen zwischen
FOUND Dateiname c	Bandanfang und gesuchter Datei aufgelistet.
FOUND Dateiname d	Meldung bekannter Dateien
FOUND	Meldung unbekannter Dateien
FOUND [Dateiname]	Meldung der gesuchten Datei
READY	Öffnen der gesuchten Datei zum Lesen

Wird die OPEN-Anweisung durch ein Programm gegeben, dann erscheinen diese Mitteilungen nicht.

Wenn die OPEN-Anweisung ausgeführt wird, um eine Banddatei zu schreiben, dann stellt der CBM-Computer folgende Mitteilung auf dem Bildschirm dar, falls am Bandgerät noch keine Bedientaste betätigt worden ist:

```
PRESS PLAY & RECORD ON TAPE #1
OK
```

Taste wurde gedrückt
das Magnetband beginnt sich zu bewegen

Nach Bedienung des Bandgeräts trägt der CBM-Computer eine Bandanfangs-Marke ein; danach stoppt die Bandbewegung. Hier einige Beispiele für die Form von OPEN-Anweisungen:

OPEN 1	Öffnen der logischen Datei 1. Da eine Gerätenummer fehlt, wird Kassette #1 angenommen (Ersatzwert =1). Da die Sekundäre Adresse fehlt, wird eine Leseoperation angenommen (Ersatzwert = 0). Da ein Dateiname fehlt, wird aus der ersten Banddatei gelesen.

OPEN 1,1	Wie oben, da der zweite Parameter ein Ersatzwert ist.
OPEN 1,1,0	Wie oben, da der zweite und dritte Parameter Ersatzwerte sind.
OPEN 1,1,0, "DAT"	Wie oben, Zugriff erfolgt jedoch auf die Datei "DAT".
OPEN 3,1,2	Öffnen der logischen Datei 3 für Bandgerät #1. Schreiben in eine neue Datei mit Bandende-Marke. Die neue Datei erhält keinen Namen.
OPEN 3,1,2,"PENTAGRAM"	Wie oben, Dateiname ist jedoch "PENTAGRAM"

SCHLIESSEN EINER DATEI AUF BANDKASSETTE : CLOSE

Da Öffnen und Schließen einer Datei eine logische Einheit bilden, werden wir der Klarheit wegen vor Beschreibung des Zugriffs zu einer Datei das Schließen einer Datei behandeln. Behalten Sie aber in Erinnerung: das Schließen einer Datei mit der Anweisung CLOSE darf erst am Ende eines Dateizugriffs erfolgen. Sie können auf keine Datei zugreifen, die bereits mit CLOSE geschlossen wurde.

Folgende Anweisung schließt eine Datei:

```
CLOSE N
```

Der Parameter N ist die logische Dateinummer, die als erster Parameter in der vorangegangenen OPEN-Anweisung erschienen ist.

Nach dem Lesen von einer Banddatei wird mit der Anweisung CLOSE jeder weitere Lesevorgang verhindert. Es bleibt ohne Folgen, wenn Sie nach dem Lesen von einer Banddatei die CLOSE-Anweisung vergessen, aber Sie geben damit einer nachlässigen Programmierpraxis nach.

Nach dem Schreiben auf eine Banddatei muß eine CLOSE-Anweisung folgen. Erinnern wir uns, daß das Übertragen von Daten zum Bandgerät über einen Pufferspeicher erfolgt. Ist der Pufferspeicher gefüllt, dann wird sein Inhalt automatisch auf die Bandkassette geschrieben. Haben die letzten zu übertragenden Daten den Pufferspeicher jedoch nicht ganz gefüllt, dann kann nur die CLOSE-Anweisung bewirken, daß sie auf die Bandkassette geschrieben werden. Wenn die Datei aus irgendeinem Grund nicht geschlossen wird, dann werden diese Restinhalte des Pufferspeichers nicht ausgeschrieben, und das kann Probleme verursachen. Außerdem wird mit der CLOSE-Anweisung nach dem Schreiben auf eine Bandkassette die Dateiende-Marke auf Bandkassette geschrieben. Der Computer benötigt diese Dateiende-Marke, um die Dateien voneinander unterscheiden zu können. Ohne eine Dateiende-Marke würde der Computer die folgende Datei lesen, als ob sie Teil der vorangegangenen sei, und hierdurch entstehen sicherlich Fehler.

Wenn Sie eine Datei schließen, die mit einer Sekundären Adresse 2 in der OPEN-Anweisung geöffnet wurde, dann wird seine Bandende-Marke auf die Bandkassette geschrieben. Eine Bandende-Marke meldet dem CBM-Computer, daß sich keine weiteren Dateien auf der Bandkassette befinden; die Bandende-Marke dient aber nicht dazu, anzugeben, daß das physikalische Ende des Magnetbandes erreicht wurde. Fehlt die Bandende-Marke, dann setzt der CBM-Computer beim nächsten Lesevorgang eine Suche nach Dateien über die Grenzen der Dateiaufzeichnungen hinaus fort, wobei er frühere Aufzeichnungen auf dem Band finden kann, die dann fälschlicherweise als gültige Daten interpretiert werden.

Sie müssen jedoch nicht jede einzelne Datei mit einer CLOSE-Anweisung schließen. Mit der END-Anweisung am Programmende werden automatisch alle offenen Dateien auf der Bandkassette geschlossen. Warum sich also um das Schließen einzelner Dateien kümmern? Hierfür gibt es zwei Gründe:

1. CLOSE-Anweisungen am Ende jeder Datei verbessern den logischen Aufbau eines Programms und verringern daher Programmierfehler.
2. Es dürfen maximal 10 Dateien auf Bandkassette gleichzeitig offen sein.

Nur wenige Programme benötigen mehr als 10 offene Dateien auf einer Bandkassette. Wenn Sie aber die Mühe nicht scheuen, Dateien nach der Benutzung zu schließen, dann kann auch das Programm entlastet werden von Dateien, die nicht mehr benötigt werden. Besonders in langen Programmen, die in kleineren Abschnitten (Modul) geschrieben wurden, können unabgeschlossene Dateien Probleme verursachen. Wenn jeder Modul einige Dateien offenläßt, dann sind schnell 10 offene Dateien erreicht, und die elfte OPEN-Anweisung löst eine Fehlermeldung aus. Diese Art von Fehlern sind äußerst schwierig aufzufinden, da sie plötzlich in einem Programm auftauchen, das vorher fehlerfrei gelaufen sein kann.

Es kostet sehr wenig Programmieraufwand oder Programmlaufzeit, jede Datei individuell mit einem CLOSE abzuschließen, nachdem auf sie zugegriffen wurde. Und Sie können damit spätere Fehlermeldungen vermeiden.

Die CLOSE-Anweisung kann entweder in direktem Dialog mit dem Computer oder in einem Programmablauf ausgeführt werden. Soll eine Banddatei beschrieben werden, ohne daß vorher die Bedientasten am Bandgerät betätigt worden sind, dann stellt der CBM-Computer anläßlich einer CLOSE-Anweisung folgende Mitteilung auf dem Bildschirm dar:

```
PRESS PLAY & RECORD ON TAPE #1 <-- Taste drücken
OK <------------------------------ Magnetband bewegtsich;- Pufferspeicher wird geladen
```

Bedientasten am Bandgerät müssen nicht betätigt werden, wenn ein Lesevorgang mit READ durch CLOSE abgeschlossen wird. Hier einige Beispiele für die Form von CLOSE-Anweisungen:

```
10  CLOSE 1        logische Datei 1 schließen
100 CLOSE 14       logische Datei 14 schließen
210 A=14
220 CLOSE A        logische Datei A schließen
```

ZUGRIFF AUF BANDKASSETTEN-DATEIEN

Nach Offnen einer Banddatei können Sie entweder von der Datei lesen oder auf die Datei schreiben. Die sekundäre Adresse in einer OPEN-Anweisung entscheidet über den zulässigen Zugriff. Zugriffe können fortgesetzt werden, bis die Datei mit einer CLOSE-Anweisung geschlossen wurde. Bedenken Sie aber, daß das Lesen von einer Datei auf Bandkassette oder das Schreiben in eine Datei auf Bandkassette ein sequentieller Vorgang ist. Die erste Bandaufzeichnung stellt immer die erste Datei auf einem Band dar. Wenn Sie die zehnte Datei auf einer Bandkassette lesen wollen, dann müssen Sie zuerst die Aufzeichnungen 1 bis 10 lesen. Wie Sie umgekehrt nicht eine zehnte Datei aufzeichnen, ohne zuerst die Dateien 1 - 9 aufgezeichnet zu haben.

Sie haben sicherzustellen, daß jedes Bandgerät mit der richtigen Kassette geladen ist, auf die ein ausführendes Programm zugreifen soll.

Wenn Sie nur über ein Bandgerät verfügen, dann ist es das sicherste Verfahren, das Bandgerät zunächst mit der Programm-Kassette zu laden und das gewünschte Programm in den Arbeitsspeicher des Computers zu lesen, und danach die Programm-Kassette gegen die Daten-Kassette auszutauschen, mit der das Programm zu arbeiten hat. Bei zwei Bandgeräten stellen Sie sicher, daß Bandkassetten und Bandgeräte richtig einander zugeordnet wurden. Nach dem Laden eines Programms in den Arbeitsspeicher des Computers können Sie die Bandkassette mit dem gespeicherten Pro-

gramm entfernen, wenn Sie wollen, und wenn das Programm die Verfügung über das Bandgerät erlaubt.

Vor dem ersten Ansprechen einer Bandkassette durch den Computer sollten keine Bedientasten am Bandgerät betätigt werden. Den Zeitpunkt der Tasteneingabe teilt der CBM-Computer über eine Nachricht auf dem Bildschirm mit.

Erinnern Sie sich daran, daß es in der Verantwortung des Benutzers liegt, das Magnetband innerhalb der Bandkassette in die richtige Anfangsstellung zu bringen. Das Bandgerät beginnt seine Aufzeichnungen an der augenblicklichen Anfangsstellung des Magnetbandes. Zum Lesen von der Bandkassette beginnt das Bandgerät seinen Suchvorgang nach einer Datei an der augenblicklichen Anfangsstellung des Magnetbandes, so daß vor der Anfangstellung stehende Dateien nicht gefunden werden.

Das Speichern von Daten auf Bandkassette wird mit einer PRINT#-Anweisung veranlaßt:

PRINT #f,data

wobei:

f = logische Dateinummer (Bereich 1-255) in Übereinstimmung mit den f-Nummern in OPEN und CLOSE

data = zu speichernde Daten

PRINT# kann nicht als (?) eingegeben werden.

Daten zur Bandaufzeichnung werden mit der PRINT#-Anweisung in einen Teil des Arbeitsspeichers gelesen, der im Computer als Pufferspeicher zum Bandgerät dient. Der Pufferspeicher ist nach 191 Byte Daten voll, die dann als geschlossener Block auf die Bandkassette geschrieben werden. Ein solcher Block kann einen Teil einer Aufzeichnung, eine vollständige Aufzeichnung oder mehrere Aufzeichnungen enthalten.

Mit der PRINT#-Anweisung können sowohl Zahlen wie auch Zeichenketten auf Bandkassette geschrieben werden.

SPEICHERN VON ZAHLEN AUF BANDKASSETTE : PRINT#

Wenn Zahlen auf Magnetband geschrieben werden, muß jeder Zahl ein Wagenrücklauf (RETURN) folgen.

Wir zeigen Ihnen mit dem folgenden Programm NUM. PRINT#, wie Zahlen auf eine Bandkassette geschrieben werden. Das Programm schreibt die Zahlen 1 - 10 auf Bandkassette.

Zuerst stellt das Programm auf dem Bildschirm eine Nachricht dar, in der es seinen Anwendungszweck mitteilt und Startanweisungen an den Benutzer gibt:

```
10 PRINT"■**ANLEGEN EINER DATEI FUER ZAHLEN**":PRINT
20 PRINT"**KASSETTE LADEN-DANACH RETURN EINGEBEN**":PRINT
30 GET A$:IF A$="" THEN 30
```

In Zeile 20 wird der Benutzer angewiesen, eine Kassette in das Bandgerät einzulegen, auf den Anfang des Magnetbandes zurückzuspulen und danach die Taste RETURN zu drücken. In Zeile 30 wartet das Programm auf eine Tasteneingabe. Mit dieser Warteschleife gibt das Programm dem Benutzer Zeit für die gefragten Vorbereitungen des Bandgeräts.

Die Wartestellung des Programms in Zeile 30 könnte auch unbeabsichtigt, etwa durch eine Ellbogenberührung des Tastenfelds beendet werden. Außerdem gilt die Anweisung an den Benutzer, nach Abschluß seiner Arbeiten die Taste RETURN zu bedienen, so daß man die Warteschleife besser wie folgt formulieren sollte:

```
30 GET A$:IF A$<>CHR$(13) THEN 30
```

Sobald die RETURN-Taste bedient wurde, geht das Programm zur nächsten Zeile über, auf der mit einer OPEN-Anweisung eine Banddatei geöffnet wird:

```
40 PRINT"**OEFFNEN EINER DATEI FUER ZAHLEN**":OPEN1,1,2"ZAHLEN"
```

Diese OPEN-Anweisung öffnet eine Datei mit der logischen Dateinummer 1, wählt als externes Gerät mit der Adressnummer 1 das Bandgerät aus und bestimmt mit der Sekundären Adresse 2, daß die Datei zum Schreiben geöffnet und am Dateiende eine Bandende-Marke geschrieben wird. Die Daten-Datei trägt den Namen ZAHLEN.

Als nächstes richten wir eine FOR-NEXT-Schleife zur Bildung der Zahlen 1 - 10 ein, die auf dem Bildschirm dargestellt und auf die Bandkassette gespeichert werden:

```
50 FOR N=1 TO 10
60 PRINT N          <-- N auf dem Bildschirmdarstellen
70 PRINT#1,N        <-- N in der Datei# 1 ("ZAHLEN") speichern
80 NEXT N
```

Zeile 60 enthält den PRINT-Befehl für die Darstellung auf dem Bildschirm, Zeile 70 den PRINT-Befehl für das Schreiben auf Band. Erinnern Sie sich an die Form, die der PRINT-Befehl für das Speichern auf Bandkassette haben muß:

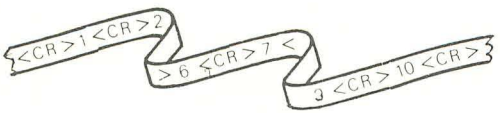
falsch	**richtig**
?#1,N	PRINT#1,N
PRINT N	
PRINT #1,N	
PRINT#1N	
PRINT1,N	

Jede der hier aufgeführten falschen Formen eines PRINT-Befehls führt zu einer Fehlermeldung (SYNTAX-Fehler), außer PRINT N, die zur Darstellung von N auf dem Bildschirm führt.

Wenn alles richtig abläuft, werden mit Zeilen 50 bis 80 die Zahlen 1 bis 10 gleichzeitig auf dem Bildschirm dargestellt und auf Band gespeichert:

Bildschirm **Magnetband**

1
2
3
4
5
6
7
8
9
10



CR = Wagenrücklauf (RETURN)

Während die Anweisung PRINT N den Wagenrücklauf in Form eines Zeilensprungs auf dem Bildschirm tatsächlich ausführt, schreibt die Anweisung PRINT# den Wagenrücklauf in Form seiner CODE-Nummer auf das Band. In Zeile 70 wird daher nach jeder Ausgabe einer Zahl N der Kode für Wagenrücklauf auf das Band ge-

schrieben, während in Zeile 60 nach jeder Zahlenausgabe ein Zeilensprung stattfindet. Achten Sie daher beim Beschreiben von Bandkassetten mit Zahlen auf die Grammatik der PRINT#-Anweisung : PRINT " schreibt "den Wagenrücklauf (RETURN) auf das Band,PRINT dagegen führt ihn aus.

Die Speicherung der Zahlen auf Bandkassette wird mit der Anweisung CLOSE abgeschlossen. Das Schließen der Datei wird auf dem Bildschirm mitgeteilt:

```
90 PRINT"**SCHLIESSEN DER DATEI FUER ZAHLEN**":CLOSE1
100 END
```

Die Anweisungen OPEN und CLOSE müssen die gleiche logische Dateinummer verwenden:

```
OPEN    1,1,2,"ZAHLEN"
.
.
.
CLOSE   1
```

Hier nun die vollständige Programmauflistung für **NUM.PRINT #**

```
10 PRINT"**ANLEGEN EINER DATEI FUER ZAHLEN**":PRINT
20 PRINT"**KASSETTE LADEN-DANACH RETURN EINGEBEN**":PRINT
30 GET A$:IF A$="" THEN 30
40 PRINT"**OEFFNEN EINER DATEI FUER ZAHLEN**":OPEN1,1,2"ZAHLEN"
50 FOR N=1 TO 10
60 PRINT N
70 PRINT#1,N
80 NEXT N
90 PRINT"**SCHLIESSEN DER DATEI FUER ZAHLEN**":CLOSE1
100 END
```

Und hier ein Programmlauf:

```
** ANLEGEN EINER DATEI FUER ZAHLEN**

**KASSETTE LADEN-DANACH RETURN EINGEBEN**

**OEFFNEN EINER DATEI FUER ZAHLEN**

PRESS PLAY & RECORD ON TAPE #1
OK

1
2
3
4
5
6
7
8
9
10
** SCHLIESSEN DER DATEI FUER ZAHLEN**
```

SPEICHERN VON ZEICHENKETTEN AUF BANDKASSETTE : PRINT#

Anders als bei Zahlen können Zeichenketten mit einem Komma oder Wagenrücklauf als Trennsymbol auf Bandkassette geschrieben werden. Die Wirkung der bei-

den Trennsymbole ist verschieden und zeigt sich erst, wenn die Zeichenketten wieder von der Bandkassette gelesen werden. Die Anweisung INPUT# zum Lesen einer Bandkassette liest sämtliche Zeichenketten-Variablen, die sie bis zum nächsten Wagenrücklauf-Symbol auf der Bandkassette vorfindet. Kommas haben also nur die Funktion, Trennsymbol zwischen den einzelnen Zeichenketten-Variablen zu sein, die mit einer INPUT#-Anweisung gruppenweise eingelesen werden. Die letzte Zeichenketten-Variable, die von einer INPUT#-Anweisung gelesen werden soll, muß mit einem Wagenrücklauf-Symbol abgeschlossen werden.

Besondere Programmiertechniken sind erforderlich, um Zeichenketten-Variablen mit Kommas zu trennen. Überdies kann der abwechselnde Gebrauch von Komma und Wagenrücklauf als Trennsymbol selbst für erfahrene BASIC-Programmierer verwirrend sein. Befolgen Sie daher aufmerksam die folgenden Programmbeispiele, ehe Sie eigene Programme versuchen.

Wir werden das Programm NUM.PRINT# derart abwandeln, daß es die Worte "EINS" bis "ZEHN" als Zeichenketten auf Band schreibt. Das neue Programm trägt den Namen WORD.PRINT#. Die Worte können entweder mit der Anweisung INPUT oder READ/DATA eingegeben werden. Unser Beispielprogramm macht Gebrauch von READ/DATA-Anweisungen. Zeile 60 enthält eine READ-Anweisung als Teil der FOR-NEXT-Schleife. Die zugehörige DATA-Anweisung steht am Programmende. Und nun die Auflistung der endgültigen Form des Programms und das Beispiel eines Programmlaufs:

WORD. PRINT #

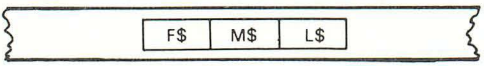
```
10 PRINT"**ANLEGEN EINER DATEI FUER TEXT**":PRINT
20 PRINT"**KASSETTE LADEN-DANACH RETURN EINGEBEN**"
30 GET A$:IF A$="" THEN 30
40 PRINT"**OEFFNEN EINER DATEI FUER TEXT**":OPEN1,1,2"ZAHLWORTE":PRINT
50 FOR N=1 TO 10
60 READ N$
70 PRINT N$
80 PRINT#1,N$
90 NEXT N
100 PRINT"**SCHLIESSEN DER DATEI FUER TEXT**":CLOSE1
110 DATA EINS,ZWEI,DREI,VIER,FUENF,SECHS,SIEBEN,ACHT,NEUN,ZEHN
120 END
**ANLEGEN EINER DATEI FUER TEXT**

**KASSETTE LADEN-DANACH RETURN EINGEBEN**

**OEFFNEN EINER DATEI FUER TEXT**

PRESS PLAY & RECORD ON TAPE #1
OK

EINS
ZWEI
DREI
VIER
FUENF
SECHS
SIEBEN
ACHT
NEUN
ZEHN
**SCHLIESSEN DER DATEI FUER TEXT**
```

Jede Übertragung einer Zeichenkette auf die Bandkassette schließt das Programm mit einem Wagenrücklauf-Symbol (RETURN) ab.

Lassen Sie uns nun sehen, wie Kommas zur Trennung von Zeichenketten-Variablen dienen, die auf Bandkassette geschrieben werden. Um Kommas als Trennsymbol zu verwenden, genügt es nicht, die Variablen in der Parameter-Liste eines PRINT-Befehls durch Komma zu trennen. Ein Beispiel:

```
10 PRINT#1,F$,M$,L$
```

Wenn diese Anweisung ausgeführt wird, wird der Inhalt der drei Zeichenketten-Variablen F$, M$ und L$ zu einer einzigen Zeichenketten-Variablen verkettet und wie folgt auf der Bandkassette gespeichert:



Ein Komma in der Rolle eines Trennsymbols zwischen Zeichenfeldern kann mit einer der beiden folgenden Methoden eingefügt werden:

1. Darstellen des Trennsymbols zwischen Anführungszeichen

```
PRINT#1,F$;","M$;",";L$
```

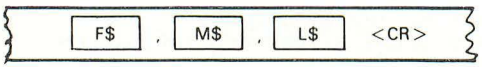
2. Gebrauch der Funktion CHR$

```
PRINT#1,F$;CHR$(44);M$;CHR$(44);L$
```

Trennsymbole

CHR$ (44) ist die Darstellung des Komma-Zeichens mit Hilfe der Funktion CHR$.

Hier eine bildliche Darstellung, wie F$, M$ und L$ nach Speicherung auf der Bandkassette durch Kommas getrennt sind, die zwischen F$ — M$ und M$ — L$ stehen:



Das folgende Programm namens NAMES.PRINT# erreicht durch Komma-Trennsymbole, daß die Zeichenfelder eines Namens F$, M$ und L$ (FIRST, MIDDLE, LAST) getrennt sind :

NAMES. PRINT #

```
10 PRINT"■** ANLEGEN EINER NAMEN-DATEI **":PRINT
20 PRINT"** KASSETTE LADEN-DANACH RETURN EINGEBEN **"
30 GET A$:IF A$="" THEN 30
40 PRINT"** OEFFNEN DER DATEI **":OPEN1,1,2,"NAME":PRINT
50 FOR J=1 TO 4
60 INPUT F$,M$,L$
70 PRINT F$,M$,L$
80 PRINT#1,F$;CHR$(44);M$;CHR$(44);L$
90 NEXT J
100 PRINT"** SCHLIESSEN DER DATEI **":CLOSE 1
110 END
```

Das nun folgende Programmbeispiel zeigt die Speicherung von Adresslisten auf einer Bandkassette. Das neue Programm MAIL.PRINT# speichert eine Adressenliste mit dem Namen ADRESSEN auf Bandkassette. Die Adressenliste kann dann mit dem Programm MAIL.INPUT# von der Bandkassette gelesen werden. In diesem Pro-

grammbeispiel wollen wir Programmschritte beschreiben, die zum Schreiben in Bandkassetten erforderlich sind. Bei dieser Gelegenheit wollen wir nicht den besten Programmentwurf für Dateneingaben demonstrieren. Das in Kapitel 5 beschriebene Adresslisten-Programm stellt einen guten Programmentwurf für Dateneingaben dar. Das jetzt folgende Programm für Adresslisten verwendet eine sehr einfache und nicht ausreichende Programm-Logik, aber seine Kürze und Übersichtlichkeit gestattet, die ganze Aufmerksamkeit auf die Beschreibung des Arbeitens mit Bandkassetten zu richten.

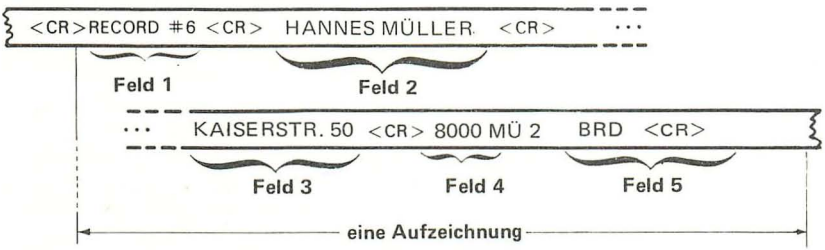
Das Adresslisten-Programm unterteilt jede Aufzeichnung von Namen und Adresse in folgende fünf Felder:

1. Aufzeichnungs-Nummer
2. Name
3. Straße
4. Stadt
5. Land

An einem Beispiel sieht das wie folgt aus:

```
** RECORD #6 **    Feld 1
                                  }
HANNES MUELLER     Feld 2         }
KAISERSTR.50       Feld 3         }  eine Aufzeichnung
8000 MUENCHEN 2    Feld 4         }
BRD                Feld 5         }
```

Dies ist natürlich die Darstellung auf dem Bildschirm, nicht auf der Bandkassette. Von der Darstellung auf der Bandkassette kann man sich folgendes Bild machen:



Hier die Auflistung des vollständigen Programms MAIL.PRINT#. Geben Sie dieses Programm in Ihren CBM-Computer und speichern Sie es auf einer Bandkassette. Dann lassen Sie sich das Programm auflisten:

MAIL.PRINT #

```
10 PRINT"■♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠"
20 PRINT"♠                              ♠"
30 PRINT"♠ ADRESSLISTEN-PROGRAMM        ♠"
40 PRINT"♠                              ♠"
50 PRINT"♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠"
60 PRINT"■■♠♠KASSETTE LADEN-DANACH RETURN EINGEBEN♠♠"
70 GET A$:IF A$="" THEN GOTO 70
80 PRINT"■♠♠OEFFNEN DER ADRESS-DATEI♠♠":OPEN1,1,2,"ADRESSEN"
85 I=0
90 I=I+1
100 PRINT"■   ♠♠ ADRESS-EINGABE NR ";I;" ♠♠"
110 PRINT"      ──────────────────      "
120 PRINT"      (ENDE DER EINGABEN MIT: 1.NAME  ?END)"
130 PRINT"■■■"
135 INPUT "1. NAME           ";NM$
```

```
140 IF NM$="END" THEN CLOSE1:PRINT "□";"♠♠PROGRAMM-ENDE♠♠":END
150 INPUT "2. STRASSE        ";SR$
160 INPUT "3. PLZ STADT PLZ ";ST$
170 INPUT "4. LAND           ";LD$
180 INPUT "■■■■■■■■■ENDERN=FELD# EINGEBEN ** SPEICHERN=0 EINGEBEN";X
190 IF X=0 THEN 220
200 IF X>=1 AND X<=4 THEN GOSUB 280
210 GOTO 180
220 PRINT#1,I
230 PRINT#1,NM$
240 PRINT#1,SR$
250 PRINT#1,ST$
260 PRINT#1,LD$
270 PRINT"■■■■■■■■■":GOTO 90
280 PRINT:ON X GOTO 290,300,310,320
290 INPUT             "1. NAME           ";NM$:RETURN
300 PRINT      :INPUT "2. STRASSE        ";SR$:RETURN
310 PRINT"■"   :INPUT "3. PLZ STADT PLZ ";ST$:RETURN
320 PRINT"■■"  :INPUT "4. LAND           ";LD$:RETURN
```

Die ersten fünf Zeilen (10 - 50) geben eine Kurzbeschreibung der Programmanwendung. Der nächste Programmabschnitt (Zeilen 60 und 70) gibt dem Benutzer die Anweisung, eine Bandkassette einzusetzen.

Die OPEN-Anweisung in Zeile 80 öffnet eine Datei:

```
80 PRINT"■■♠♠OEFFNEN DER ADRESS-DATEI♠♠":OPEN1,1,2,"ADRESSEN"
```

Die Datei ADRESSEN wird unter der logischen Dateinummer 1 auf der Bandkassette geöffnet und mit einem Bandendezeichen abgeschlossen. Vor der eigentlichen OPEN-Anweisung wird auf dem Bildschirm die Nachricht "ÖFFNEN DER ADRESS-DATEI" dargestellt. Diese Nachricht wird dem Benutzer Sekunden vor der tatsächlichen Dateiöffnung gegeben.

Das Magnetband ist jetzt vorbereitet für die Speicherung von Daten. Daten sollen vor der Speicherung auf Band am Bildschirm sichtbar sein, so daß sie noch auf Fehler überprüft werden können. In Zeilen 130 bis 170 werden die am Tastenfeld eingegebenen Daten zur Darstellung auf dem Bildschirm eingelesen.

In Zeilen 90 und 100 dient die Variable I der fortlaufenden Zählung der Namens- und Adressaufzeichnungen; auf dem Bildschirm wird angegeben, die wievielte Aufzeichnung augenblicklich eingegeben wird.

In Zeilen 130 bis 170 werden die Variablen NM$, SR$, ST$ und LD$ als getrennte Zeichenfelder eingelesen. Das Ende jedes Zeichenfeldes wird durch einen Wagenrücklauf angezeigt. Nach Eingabe aller vier Zeichenfelder wird dem Benutzer in Zeile 180 die Möglichkeit gegeben, Zeichenfelder zu ändern oder die Eingaben zur Speicherung freizugeben.

Ist eines der Zeichenfelder falsch, dann gibt der Benutzer die zugehörige Zeichenfeld-Nummer 1 - 4 ein, woraufhin der Sprung zu einem Unterprogramm in Zeile 280 erfolgt, das der Korrektur von Zeichenfeldern dient. Der Kursor wird auf das Zeichenfeld mit der angegebenen Nummer (Variable X) geführt, womit dem Benutzer die Möglichkeit für eine Änderung gegeben ist. Danach springt das Unterprogramm auf Zeile 180 des Hauptprogramms zurück, so daß der Benutzer die Änderung eines weiteren Zeichenfeldes angeben kann. Sind alle Zeichenfelder richtig, dann gibt der Benutzer eine Ø ein und das Programm setzt in Zeile 220 fort.

Jede der Anweisungen in Zeilen 220 bis 260 schreibt ein Feld der Aufzeichnung:

```
6 <CR>HANNES MUELLER <CR>KAISERSTR. 50 <CR>8000
```

Vergewissern Sie sich, ob die logische Dateinummer in den PRINT#-Anweisungen die gleiche ist wie in der OPEN-Anweisung in Zeile 80.

Nach erfolgter Bandaufzeichnung kehrt das Programm nach Zeile 90 zurück, um die nächste Aufzeichnung vorzubereiten. Das Ende der Adresseingaben zeigt der Benutzer durch Eingabe des Namens "ENDE" an. Sobald in Zeile 140 der Name NM$ = "ENDE" festgestellt wird, wird die Banddatei geschlossen und eine Bandende-Marke geschrieben, wie in der OPEN-Anweisung von Zeile 80 vorgesehen.

Sie werden beobachtet haben, daß sich das Magnetband nicht bei jeder Adresseingabe bewegt. Der CBM-Computer speichert Daten für die Bandkassette zunächst in einem Pufferspeicher. Sobald der Pufferspeicher voll ist, wird der gesamte Speicherinhalt als Block auf das Magnetband geschrieben. Ein Block kann den Teil einer Aufzeichnung, eine vollständige Aufzeichnung oder mehrere Aufzeichnungen enthalten. Der CBM-Computer trennt die einzelnen Blöcke auf dem Magnetband durch Lücken wie folgt:

Hier die Bildschirmmeldungen am Ende eines Programmlaufs:

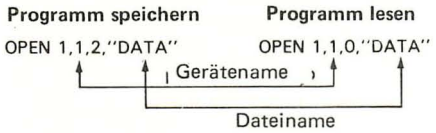
```
BREAK IN 60
READY
```

LESEN VON DATEN VON BANDKASSETTE : INPUT#, GET#

Hier die drei Programmschritte, um Daten von einer Bandkassette zu lesen:

1. Öffnen der Datei mit OPEN
2. Lesen der Datei
3. Schließen der Datei mit CLOSE

Zum Lesen muß eine Banddatei mit dem gleichen Dateinamen geöffnet werden, der schon beim Schreiben verwendet wurde. Die logische Dateinummer dagegen kann verschieden sein. Die Sekundäre Adresse muß zum Lesen den Wert 0 haben:



Zum Lesen einer Banddatei stehen zwei Anweisungen zur Verfügung: INPUT# und GET#. Die INPUT#-Anweisung dient zum Lesen ganzer Felder, die GET#-Anweisung liest jeweils ein einzelnes Symbol.

Das Lesen einer Datei wird mit der Anweisung CLOSE abgeschlossen. Die logische Dateinummer in CLOSE muß mit der Dateinummer der vorangegangenen OPEN-Anweisung übereinstimmen:

Ein gutes Verfahren zum Schließen einer Datei besteht in der Abfrage des Gerätestatus ST nach einer Dateiende-Marke. Jede Datei wird beim Schreiben mit einer Dateiende-Marke abgeschlossen. Beim Lesen der Dateiende-Marke nimmt die

Statusvariable ST den Zahlenwert 64 an. Sie können diese Statusvariable abfragen und die Datei mit folgender Anweisung schließen:

```
IF ST=64 THEN CLOSE 1
```

Sobald ST den Zahlenwert 64 annimmt, wird die Datei geschlossen.

Wir haben weiter oben das Programm NUM.PRINT# dazu verwendet, die Zahlen 1 - 10 in eine Banddatei namens ZAHLEN zu schreiben. Wir werden jetzt ein Programm namens NUM. INPUT # schreiben, mit dem wir die 10 Zahlen von der Banddatei ZAHLEN lesen und auf dem Bildschirm darstellen.

Die ersten Anweisungen in NUM.INPUT# geben dem Benutzer Anweisung, das Bandgerät vorzubereiten. Diese Anweisungen stimmen mit den ersten drei Anweisungen in NUM.PRINT# überein. Zeile 30 enthält die Warteschleife, mit dem das Programm dem Benutzer Zeit zum Einlegen einer Bandkassette gibt. Nach Abschluß der Arbeiten drücken Sie die Taste RETURN, womit das Programm in der nächsten Zeile fortsetzt:

```
10 PRINT"□◆◆ LESEN EINER NUMERISCHEN BANDDATEI◆◆":PRINT
20 "◆◆KASSETTE LADEN-DANACH RETURN EINGEBEN◆◆":PRINT
30 GET A$:IF A$="" THEN 30
```

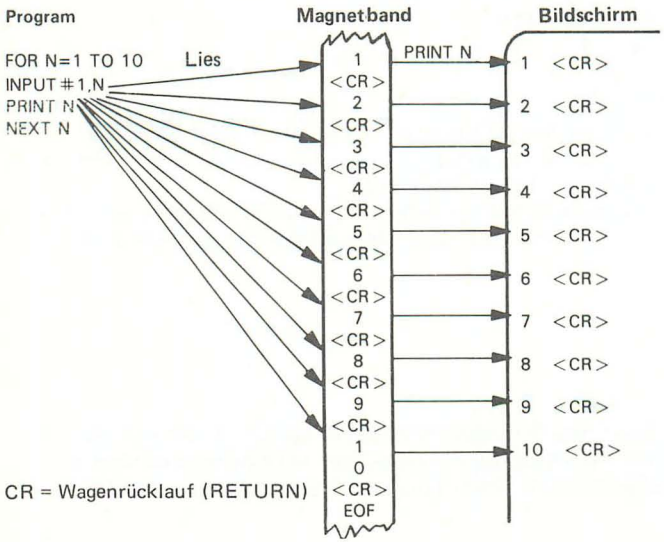
Vor dem Lesen von Daten muß zunächst die Datei geöffnet werden. Die Anweisungen in Zeile 40 öffnen die Datei# 1 im externen Gerät #1 mit der Sekundären Adresse Ø (für das Lesen der Datei) und dem Dateinamen ZAHLEN:

```
40 PRINT"◆◆OEFFNEN DER BANDDATEI◆◆":OPEN 1,1,0,"ZAHLEN":PRINT
```

Die nachfolgende FOR-NEXT-Schleife liest die ersten 10 Daten vom Band und stellt sie auf dem Bildschirm dar:

```
50 FOR I=1 TO 10
60 INPUT#1,N ←——— N vom Magnetband lesen
70 PRINT N ←——— N auf dem Bildschirm darstellen
80 NEXT I
```

Die Anweisung INPUT#1,N in Zeile 60 liest eine Zahl je Anweisungsausführung. Die FOR-NEXT-Schleife stellt die richtige Reihenfolge der Ausführungen sicher. Die Programmausführung kann wie folgt veranschaulicht werden:



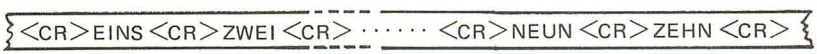
Nach dem Lesen der Daten muß die Datei geschlossen werden:

```
90 PRINT"**SCHLIESSEN DER BANDDATEI**":CLOSE1
100 END
```

Eine vollständige Auflistung des Programms NUM.INPUT# sowie das Ergebnis eines Programmlaufs werden hier gezeigt:

```
10 PRINT"■** LESEN EINER NUMERISCHEN BANDDATEI**":PRINT
20 "**KASSETTE LADEN-DANACH RETURN EINGEBEN**":PRINT
30 GET A$:IF A$="" THEN 30
40 PRINT"**OEFFNEN DER BANDDATEI**":OPEN 1,1,0,"ZAHLEN":PRINT
50 FOR I=1 TO 10
60 INPUT#1,N
70 PRINT N
80 NEXT I
85 I=0
90 PRINT"**SCHLIESSEN DER BANDDATEI**":CLOSE1
100 END
** LESEN EINER NUMERISCHEN BANDDATEI**

** KASSETTE LADEN-DANACH RETURN EINGEBEN**

** OEFFNEN DER BANDDATEI **

PRESS PLAY ON TAPE #1
OK

1
2
3
4
5
6
7
8
9
10
** SCHLIESSEN DER BANDDATEI **
```

Die Anweisung INPUT# liest ebenso Felder, die Zeichenketten-Variablen enthalten. Mit dem Programm WORD.PRINT# haben wir 10 Zeichenketten-Variablen auf Bandkassette geschrieben. Hierbei entstand die Datei ZAHLWORTE, die folgendes Aussehen hat:



Um die Datei ZAHLWORTE feldweise zu lesen, benutzen wir die Anweisung INPUT# mit einer Zeichenketten-Variablen als Parameter. Mit nur leichter Abwandlung können Sie das Programm zum Lesen numerischer Daten auf Bandkassette verwenden. Änderungen sind in Zeile 40 für den Namen der Datei und in Zeile 60 für die Variable in der Anweisung INPUT erforderlich. Die vollständige Auflistung des geänderten Programms und ein Programmlauf sehen wie folgt aus:

WORD.INPUT#

```
10 PRINT"■** LESEN DER DATEI ZAHLWORTE **":PRINT
20 PRINT"** KASSETTE LADEN-DANACH RETURN EINGEBEN **":PRINT
30 GET A$:IF A$="" THEN 30
40 PRINT"** OEFFNEN DER DATEI **":OPEN 1,1,0,"ZAHLWORTE":PRINT
50 FOR I=1 TO 10
```

```
60 INPUT#1,N$
70 PRINT N$
80 NEXT I
90 PRINT"** SCHLIESSEN DER DATEI **":CLOSE1
100 END


** LESEN DER DATEI ZAHLWORTE **

** KASSETTE LADEN-DANACH RETURN EINGEBEN **

** OEFFNEN DER DATEI **

PRESS PLAY ON TAPE #1
OK

EINS
ZWEI
DREI
VIER
FUENF
SECHS
SIEBEN
ACHT
NEUN
ZEHN
** SCHLIESSEN DER DATEI **
```

Zurück zum Programm NAMES.PRINT#: Erinnern Sie sich an die Datei NAME, in die ein Name in Form von 3 Feldern mit den Zeichenkette: F$, M$, L$ geschrieben wurde. Jede Zeichenkette bildet ein Feld, das vom folgenden Feld durch ein Komma getrennt ist. Auf dem Magnetband sieht das wie folgt aus:

HEADLY, GEORGE, JOYCE<CR>CAROL, A., SMITH<CR>

Werden die Felder nicht durch Kommas getrennt, dann werden sie wie eine durchgehende Zeichenkette gelesen und auch so auf dem Bildschirm dargestellt:

```
HEADLYGEORGEJOYCE
CAROLA.SMITH
```

Als nächstes das Programm NAME.INPUT# zum Lesen der Datei NAME. Die INPUT#-Anweisung in Zeile 60 liest alle Felder bis zum nächsten Trennsymbol, d.h. einem Wagenrücklauf. Die zwischen Wagenrückläufen liegenden Felder werden durch Kommas getrennt. Da zwischen den Wagenrückläufen 3 Felder liegen, die durch Komma getrennt sind, erscheinen auch in der INPUT#-Anweisung 3 Zeichenketten-Variablen als Parameter. Die PRINT-Anweisung in Zeile 70 stellt die 3 Zeichenketten-Variablen auf einer einzigen Zeile dar. Benachbarte Zeichenketten werden durch einen Zwischenraum getrennt. **NAME.INPUT#**

```
10 PRINT"** LESEN VON DER DATEI "CHR$(34)"NAME"CHR$(34)" **":PRINT
20 PRINT"** KASSETTE LADEN-DANACH RETURN EINGEBEN **"
30 GET A$:IF A$="" THEN 30
40 PRINT"** OEFFNEN DER DATEI **":OPEN1,1,0,"NAME":PRINT
50 FOR J=1 TO 4
60 INPUT#1,F$,M$,L$
70 PRINT F$;" ";M$;" ";L$
80 NEXT J
```

```
90 PRINT"** SCHLIESSEN DER DATEI **":CLOSE 1
100 END
** LESEN VON DER DATEI "NAME" **

** KASSETTE LADEN-DANACH RETURN EINGEBEN **

** OEFFNEN DER DATEI **

PRESS PLAY ON TAPE #1
OK

ARNOLD J. SIMPSON
BETTY S. SIMON
HANS GEORG MUELLER
CAROL ANNE SCHMIDT

** SCHLIESSEN DER DATEI **
```

Das nächste Programm zeigt, wie eine Adressliste gelesen werden kann, die vom Programm MAIL.PRINT# unter dem Dateinamen ADRESSEN auf Band geschrieben wurde. Jede Aufzeichnung besteht aus 5 Feldern: Aufzeichnungsnummer, Name, Straße, Stadt und Land. Hier das Beispiel einer einzelnen Aufzeichnung in der Datei ADRESSEN:

```
** RECORD #6  **  Feld 1

FA.HANS MUELLER     Feld 2
KAISERSTR.50        Feld 3
8 MUENCHEN 2        Feld 4
BRD                 Feld 5
```

1 Aufzeichnung, bestehend aus 5 Feldern

Nachstehend die Auflistung des Programms MAIL.INPUT#, mit dem Adresseintragungen vom Band gelesen werden können. Geben Sie MAIL.INPUT# in Ihren Computer ein und speichern Sie es auf einer Bandkassette. Dann lassen Sie sich eine Auflistung des Programms ausgeben, um der zeilenweisen Diskussion zu folgen:

MAIL. INPUT #

```
10 PRINT"■**********************************"
15 PRINT"*                                *"
20 PRINT"*                                *"
30 PRINT"* LESEN DER DATEI *ADRESSEN* *"
40 PRINT"*      MITTELS INPUT#            *"
50 PRINT"*                                *"
55 PRINT"**********************************":PRINT:PRINT
60 PRINT"** KASSETTE LADEN-DANACH RETURN EINGEBEN **■"
70 GET A$:IF A$="" THEN 70
80 PRINT"** OEFFNEN DER DATEI **":OPEN1,1,0,"ADRESSEN"
90 PRINT"■** LESEN DER DATEI **"
100 IF ST=64 THEN 9999
110 INPUT#1,I$
120 INPUT#1,NM$
130 INPUT#1,SR$
140 INPUT#1,ST$
150 INPUT#1,LD$
160 PRINT"■** RECORD #";I$;"     **"
170 PRINT"■■■■NAME    :";TAB(9);NM$
180 PRINT"STRASSE:";TAB(9);SR$
190 PRINT"STADT   :";TAB(9);ST$
200 PRINT"LAND    :";TAB(9);LD$
210 PRINT"■■■■"
220 INPUT"LESEN DER NAECHSTEN ADRESSE? J EINGEBEN";A$:IF A$="J"
                                                    GOTO 100
9999 PRINT"■** ENDE DER ADRESS-DATEI * PROGRAMM BEENDET **":CLOSE1
                                                    :END
```

Zeilen 10 bis 50 enthalten eine kurze Programmbeschreibung.

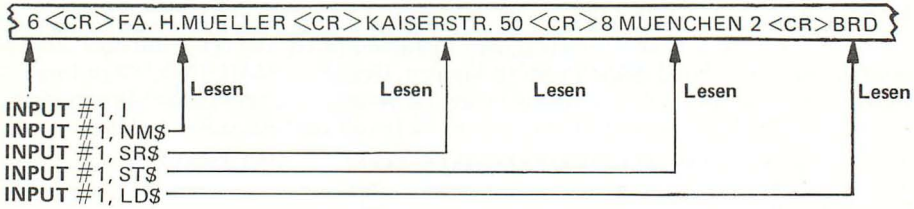
In Zeilen 60 und 70 wird der Benutzer angewiesen, eine Bandkassette einzulegen und dem Programm die Ausführung durch Drücken der Taste RETURN zu melden. Das Programm kann dann mit dem Lesen der Kundenadressen beginnen. Hierzu muß zunächst die Datei geöffnet werden.

In Zeile 80 erfolgt das Öffnen der Datei ADRESSEN, der die logische Dateinummer 1 gegeben wird und die sich auf Bandgerät #1 befindet. Zum Lesen der Datei muß man der Sekundären Adresse den Wert 0 geben:

```
80 PRINT"** OEFFNEN DER DATEI **":OPEN1,1,0,"ADRESSEN"
```

In Zeile 100 wird die Variable ST (für den Gerätestatus) nach einer Dateiende-Marke abgefragt. Wird eine Dateiende-Marke gefunden, d.h. ST = 64, dann erfolgt das Schließen der Datei in Zeile 9999. Der Status ST sollte vor jedem Lesevorgang abgefragt werden, so daß Sie nicht versuchen, Daten zu lesen, wo es keine mehr gibt.

In Zeilen 110 bis 150 erfolgt das eigentliche Lesen der einzelnen Felder durch INPUT#-Anweisungen. Jedes Feld wurde beim Schreiben auf Band durch einen Wagenrücklauf (RETURN-Symbol) abgeschlossen, so daß für jedes Feld eine eigene INPUT#-Anweisung erforderlich ist. Die Variablen I$, NM$, SR$, ST$, LD$ können beim Lesen von der Bandkassette andere Namen erhalten, als beim Schreiben auf die Bandkassette verwendet wurden. Zum Beispiel: Daten können unter dem Variablennamen X$ auf das Band geschrieben und unter dem Variablennamen A$ vom Band gelesen werden. Da diese Variablennamen weder gespeichert noch von einem Programm zum anderen übertragen werden, bleiben sie dem Computer unbekannt:



Von Band eingelesene Daten werden zunächst in einem Pufferspeicher gespeichert. Es erfolgt keine Darstellung auf dem Bildschirm, wenn sie nicht programmiert wurde.

In Zeilen 160 bis 200 wird jedes der von Band gelesenen Felder auf dem Bildschirm zur Darstellung gebracht. Die hierfür verwendeten PRINT-Anweisungen enthalten zur Textgestaltung Leerstellen und kommentierende Texte. Zeile 210 schließlich bewegt den Kursor um 4 Zeilen nach unten:

```
160 PRINT"■** RECORD #";I$;"     **"
170 PRINT"■■■■NAME    :";TAB(9);NM$
180 PRINT"STRASSE:";TAB(9);SR$
190 PRINT"STADT   :";TAB(9);ST$
200 PRINT"LAND    :";TAB(9);LD$
210 PRINT"■■■■"
```

Die Bildschirmdarstellung der Kundenadresse von vorhin sollte wie folgt aussehen:

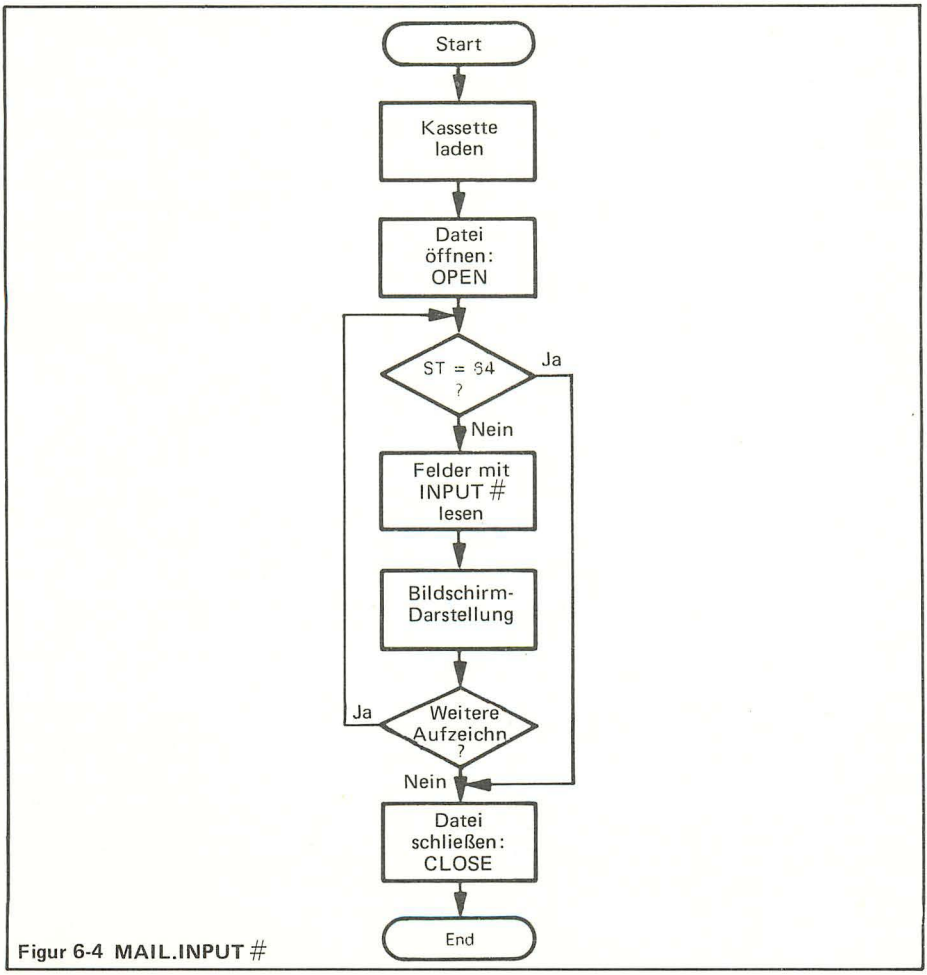
```
** RECORD #6  **

NAME    : FA.H.MUELLER
STRASSE: KAISERSTR.50
STADT   : 8 MUENCHEN 2
LAND    : BRD
```

In Zeile 220 wird der Benutzer nach erfolgter Adressendarstellung gefragt, ob eine weitere Adressdarstellung erwünscht ist:

```
220 INPUT"LESEN DER NAECHSTEN ADRESSE? J EINGEBEN";A$:IF A$="J"
GOTO 100
```

Wünscht der Benutzer eine weitere Adresseintragung zu sehen, dann springt das Programm auf Zeile 100 und wiederholt den Programmlauf, bis die Statusvariable ST das Dateiende signalisiert. Wünscht der Benutzer keinen weiteren Zugriff auf die Adressliste oder wurde das Dateiende gefunden, dann endet das Programm mit dem Schließen der Datei in Zeile 9999.

Figur 6-4 zeigt den Flußplan für das Programm MAIL.INPUT#.



Figur 6-4 MAIL.INPUT #

Hier als Beispiel die Ausgaben aus einer Adressen-Datei:

```
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠
♠                              ♠
♠                              ♠
♠ LESEN DER DATEI *ADRESSEN* ♠
♠      MITTELS INPUT#          ♠
♠                              ♠
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠


♠♠ KASSETTE LADEN-DANACH RETURN EINGEBEN ♠♠

♠♠ OEFFNEN DER DATEI ♠♠

PRESS PLAY ON TAPE #1
OK

♠♠ LESEN DER DATEI ♠♠


♠♠ RECORD #1    ♠♠
NAME    : FA.HANS MUELLER
STRASSE: KAISERSTR.50
STADT   : 8 MUENCHEN 2
LAND    : BRD

LESEN DER NAECHSTEN ADRESSE? J EINGEBEN? J


♠♠ RECORD #2    ♠♠

NAME    : FA. VAN LEEUWEN
STRASSE: WESTERBAENSTR.50
STADT   : 2513 GH DEN HAAG
LAND    : NL

LESEN DER NAECHSTEN ADRESSE? J EINGEBEN? N

♠♠ ENDE DER ADRESS-DATEI * PROGRAMM BEENDET ♠♠
```

Werden Sie nicht unruhig, wenn MAIL.INPUT# beim Programmlauf für einige Sekunden zu stoppen scheint. Beobachten Sie das Bandgerät und Sie werden sehen, daß sich das Magnetband bewegt. Was sich ereignet ist, daß der Computer die nächsten 191 Byte Daten in den Eingabespeicher liest, ehe er das Programm fortsetzt. Sobald der Pufferspeicher voll ist, wird der Computer wieder lebendig.

In Zeile 220 sehen Sie nicht das Beispiel einer guten Programmierpraxis. Diese Programmlogik veranlaßt das Lesen und Darstellen einer weiteren Kundenadresse, sobald der Benutzer die Taste J bedient. Berührt der Benutzer aber zufällig irgendeine andere Taste, dann steigt das Programm aus, d.h. es wird in Zeile 9999 beendet. Ein gut geschriebenes sieht zwei Tasteneingaben vor, z.B. "J" für Ja und "N" für Nein. Eine Bildschirmnachricht fordert den Benutzer auf, eine dieser beiden Tasten zu drücken. Jede andere Tasteneingabe sollte unterdrückt werden. Können Sie Zeile 220 derart verändern, daß das Programm in dieser Weise arbeitet?

Eine andere Möglichkeit zum Lesen einer Banddatei besteht in der Benutzung der Anweisung GET#:

GET #f,var

wobei:

f = logische Dateinummer (Bereich 1-255), übereinstimmend mit Angaben in OPEN und CLOSE

var = zu lesender Variablenname

Von GET# wird jeweils ein Symbol aus der Banddatei gelesen. Das entspricht der Anweisung GET, mit der jeweils ein Symbol vom Tastenfeld gelesen werden kann.

GET# liest Symbole, Dateiende-Marken und alles andere, was sich auf einem Magnetband befindet. Es ist besonders nützlich bei der Fehlersuche an schlechten Magnetbändern, da alle Eintragungen auf dem Magnetband gelesen werden könnnen.

GET# ermöglicht den Vergleich einzelner Symbole mit vorgegebenen Kode-Zahlen zum Zweck der Identifizierung von Symbolen.

Zwei Programmbeispiele zeigen, wie sämtliche Eintragungen einer Datei einschließlich der Dateiende-Marken gelesen und auf dem Bildschirm dargestellt werden können.

Das folgende Programm MAIL.GET#1 liest die Datei ADRESSEN symbolweise und stellt ihren Inhalt auf dem Bildschirm dar:

MAIL.GET#2

```
10 PRINT"■♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠"
15 PRINT"♠                                    ♠"
20 PRINT"♠                                    ♠"
30 PRINT"♠ LESEN DER DATEI *ADRESSEN* ♠"
40 PRINT"♠          MITTELS GET#              ♠"
50 PRINT"♠                                    ♠"
55 PRINT"♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠":PRINT:PRINT
60 PRINT"♠♠ KASSETTE LADEN-DANACH RETURN EINGEBEN ♠♠"
70 GET A$:IF A$="" THEN 70
80 PRINT"■♠♠ OEFFNEN DER DATEI ♠♠":PRINT:OPEN1,1,0,"ADRESSEN"
90 PRINT"■♠♠ ADRESSEN-DATEI ♠♠"
100 IF ST=64 THEN 9999
110 GET#1,X$
120 IF X$=CHR$(13) THEN X$="♠"
130 PRINT X$;
140 GOTO 100

9999 PRINT"■■■♠♠ ENDE DER ADRESS-DATEI * PROGRAMM BEENDET ♠♠"
10000 CLOSE 1:END
```

Zeilen 10 bis 90 entsprechen den Anfangszeilen des Programms MAIL. INPUT#. Diese Anweisungen beschreiben das Programm, geben Anweisungen für das Einlegen einer Bandkassette und öffnen die Datei.

Zeilen 100 bis 140 lesen Daten von der Datei ADRESSEN und stellen sie auf dem Bildschirm dar.

In Zeile 100 wird nach der Dateiende-Marke gefragt. Wurde sie nicht gefunden, dann wird in Zeile 110 mit der Anweisung GET# das nächste Symbol gelesen. Die GET#-Anweisung bezieht sich auf die logische Dateinummer #1 und verwendet die Variable X$ zur Speicherung der von Band gelesenen Daten. Mit dieser Anweisung erfolgt ein symbolweises Lesen der Datei.

In Zeile 120 wird der augenblickliche Inhalt X$ nach einem Wagenrücklauf, CHR$ (13), abgefragt. Stimmt der Kode in X$ mit CHR$ (13) überein, dann wird der Inhalt von X$ in den Kode für ein volles Gitternetz geändert. Siehe Tabelle 1-1 für die Darstellung und Anhang A, Tabelle A-4 für die Kode-Nummern. Durch diese Änderung wird die Ausführung des Wagenrücklaufs, d.h. der Zeilensprung auf dem Bildschirm, vermieden; mit dem Gitternetz als Symbol für einen Wagenrücklauf kann die

Datei in durchgehenden Zeilen dargestellt werden. Sehen Sie unten das Beispiel eines Programmlaufs.

In Zeile 130 ist darauf zu achten, daß die Variable in der PRINT-Anweisung mit einem Semikolon abgeschlossen wurde, da die Symbole sonst in vertikaler Anordnung auf dem Bildschirm zur Darstellung kommen.

Nach jedem Lesen und Darstellen eines Symbols von der Bandkassette fragt das Programm den Gerätestatus ST ab und liest ein weiteres Symbol. Dieser Vorgang wiederholt sich, bis ST = 64, d.h. das Dateiende erreicht ist. Die Aufgabe von MAIL.GET#1 ist abgeschlossen, wenn in Zeile 100 das Dateiende festgestellt wurde. Das Programm endet in Zeile 9999 mit dem Schließen der Datei. Hier das Beispiel eines Programmlaufs mit MAIL.GET#1 und der Datei ADRESSEN.

```
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠
♠                                  ♠
♠                                  ♠
♠ LESEN DER DATEI *ADRESSEN* ♠
♠         MITTELS GET#             ♠
♠                                  ♠
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠

♠♠ KASSETTE LADEN-DANACH RETURN EINGEBEN ♠♠

♠♠ OEFFNEN DER DATEI ♠♠
PRESS PLAY ON TAPE #1
OK
♠♠ ADRESSEN-DATEI ♠♠

 1 ▒ACME MANUFACTURING CO.▒1235 MAIN ST.
▒DOWNTOWN▒IL    62501▒ 2 ▒BENJAMIN FRANKL
IN▒12 LIBERTY TOWER▒PHILADELPHIA    16524
▒ 3 ▒NEIL ARMSTRONG▒597 SEA OF TRANQUILI
TY▒EARTHVIEW▒LUNAR    00000▒ 4 ▒MAMMOTH D
ISTRIBUTION CO.▒INDUSTRIAL PARK▒CITY OF
INDUSTRY▒CA    92425▒ 5 ▒HENRY MUSCATEL▒3
19 OAK ST.▒NAPA▒CA    95303▒ 6 ▒WIDGET SU
PPLY CO.▒555 BOGUS AVE.▒GERTIE▒TENNESSEE
    38901▒

♠♠ ENDE DER ADRESS-DATEI * PROGRAMM BEENDET ♠♠
```

Das folgende MAIL.GET#2 liest die Datei ADRESSEN und stellt jede Aufzeichnung getrennt auf dem Bildschirm dar. Hier die Auflistung des Programms:

MAIL .GET#2

```
10 PRINT"■♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠"
15 PRINT"♠                                  ♠"
20 PRINT"♠                                  ♠"
30 PRINT"♠ LESEN DER DATEI *ADRESSEN* ♠"
40 PRINT"♠          MITTELS GET#            ♠"
50 PRINT"♠                                  ♠"
55 PRINT"♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠":PRINT:PRINT
60 PRINT"♠♠ KASSETTE LADEN-DANACH RETURN EINGEBEN ♠♠":PRINT
70 GET A$:IF A$="" THEN 70
80 PRINT"♠♠ OEFFNEN DER DATEI ♠♠":PRINT:OPEN1,1,0,"ADRESSEN"
90 PRINT:PRINT"■♠♠ ADRESSEN-DATEI ♠♠":PRINT
95 F=0:R=0:N=0
100 IF ST=64 THEN 9999
110 GET#1,X$
120 IF X$=CHR$(13) THEN F=F+1
130 PRINT X$;
```

```
140 IF F=5 THEN GOSUB 160
145 IF N=1 THEN GOTO 9999
150 GOTO 100
160 PRINT
170 R=R+1
180 IF R>2 THEN PRINT"WEITERE ADRESSEN? J ODER N EINGEBEN:";
184 INPUT A$
185 IF A$="J" THEN R=0
186 IF A$="N" THEN N=1
190 F=0:A$="":PRINT:RETURN
9999 PRINT"■■■■♠♠ PROGRAMM BEENDET ♠♠":CLOSE1:END
```

Geben Sie das Programm MAIL.GET#2 in Ihren Computer ein und speichern es auf einer Bandkassette. Dann lassen Sie sich eine Auflistung ausgeben.

Die Zeilen 10 bis 100 im Programm MAIL.GET#2 sind identisch mit denen im Programm MAIL.GET#1. Dieser Teil des Programms unterrichtet den Benutzer über die Funktion des Programms, gibt Anweisungen an den Benutzer und öffnet die Datei ADRESSEN in Vorbereitung des folgenden Lesevorgangs.

Der Unterschied zwischen den Programmen MAIL.GET#2 und MAIL.GET#1 liegt in Zeile 120. Statt beim Lesen eines Wagenrücklaufs, d.h. wenn X$ = CHR$(13), den Inhalt von X$ in den Kode für ein volles Gitternetz zu ändern, wird ein Wagenrücklauf-Zähler in Form der Variablen F um den Zahlenwert 1 erhöht.

Das Programm MAIL.PRINT# hatte beim Schreiben der Banddatei jedes Feld mit einem Wagenrücklauf markiert. Jede Aufzeichnung einer Adresse bestand aus fünf Feldern. Das Programm MAIL.GET#2 zählt diese Felder. In Zeile 140 wird erkannt, ob bereits fünf Felder, d.h. fünf Wagenrückläufe gelesen wurden. Daraufhin erfolgt der Sprung in ein Unterprogramm.

Zeile 160 fügt eine Leerzeile zwischen zwei Aufzeichnungen ein.

In Zeile 170 dient die Variable R als Zähler für die bereits von der Datei gelesenen Adressen. Sind mehr als zwei Adressen gelesen worden, d.h. liegen drei Adressen vor, dann ist der Bildschirm voll und der Benutzer wird in Zeile 180 gefragt, ob er einen neuen Adressensatz wünscht. Wenn ja, werden der Adressenzähler R und der Wagenrücklauf-Zähler F auf den Anfangswert 0 zurückgesetzt und in Zeile 100 das Lesen von drei weiteren Adressen begonnen. Das geht solange, bis der Benutzer etwas anderes als ein J für Ja eingibt oder der Gerätestatus ST = 64 meldet; das Programm wird dann durch Schließen der Datei beendet.

Figur 6-5 zeigt den Flußplan des Programms MAIL.GET#2.

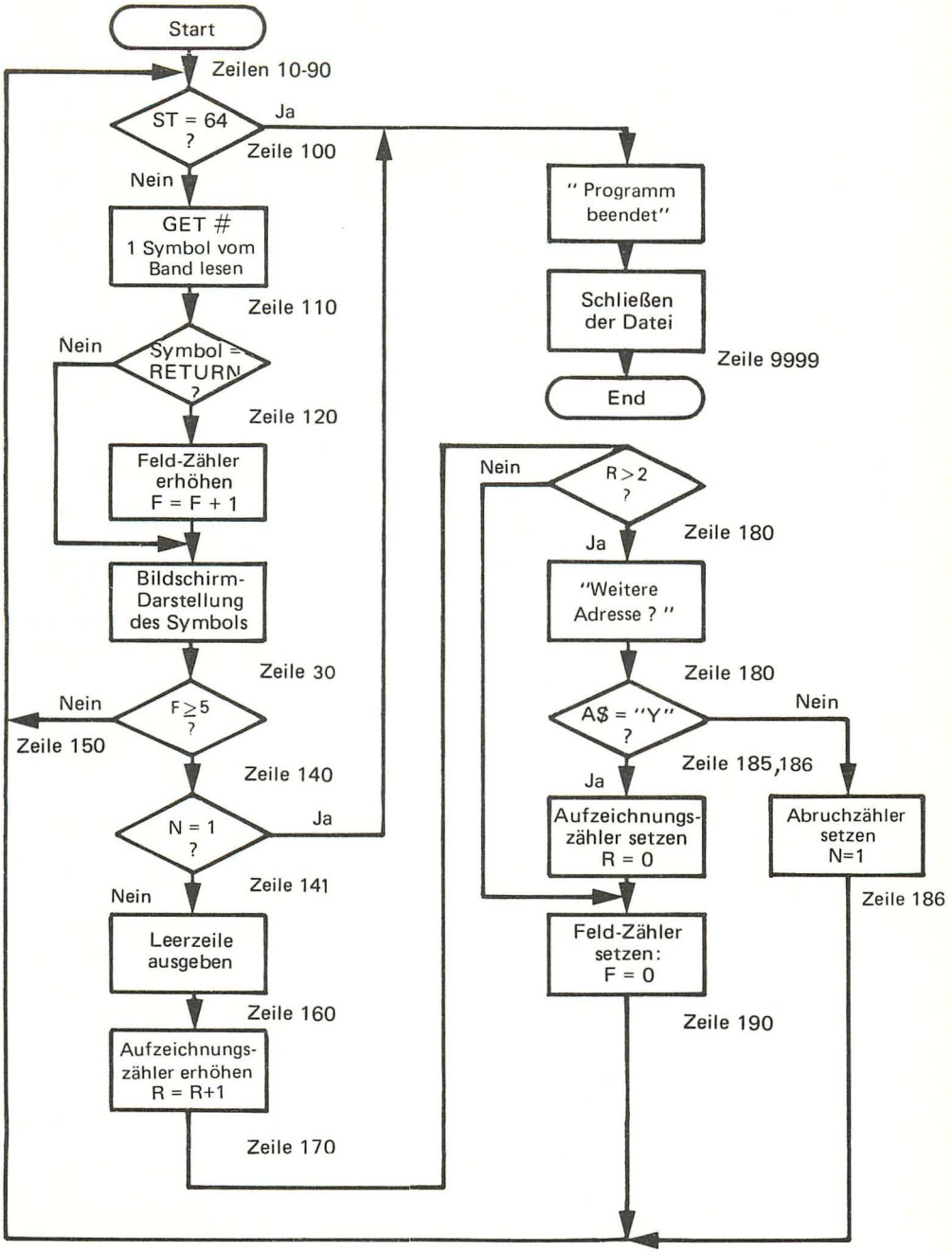
Obwohl zwischen GET# und INPUT# Ähnlichkeiten bestehen, ist eine Textgestaltung mit GET# schwieriger, wenn kommentierende Texte, Einrückungen oder Zwischenräume benötigt werden.

Ähnlich wie X$ mit CHR$ (13) verglichen wurde, können auch andere Trennsymbole zwischen Feldern abgefragt werden, um eine Textgestaltung auf dem Bildschirm zu erreichen.

Hier das Beispiel für einen Programmlauf mit MAIL.GET#2 und der Datei ADRESSEN:

```
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠
♠                             ♠
♠                             ♠
♠ LESEN DER DATEI *ADRESSEN* ♠
♠         MITTELS GET#        ♠
♠                             ♠
♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠♠
♠♠ KASSETTE LADEN-DANACH RETURN EINGEBEN ♠♠

♠♠ OEFFNEN DER DATEI ♠♠
```

```
PRESS PLAY ON TAPE #1
OK

** ADRESSEN-DATEI **

 1
ACME MANUFACTURING CO.
1235 MAIN ST.
DOWNTOWN
IL   62501


 2
BENJAMIN FRANKLIN
12 LIBERTY TOWER
PHILADELPHIA
PA   16524


 3
NEIL ARMSTRONG
597 SEA OF TRANQUILITY
EARTHVIEW
LUNAR   00000

WEITERE ADRESSEN? J ODER N EINGEBEN:?J
 4
MAMMOTH DISTRIBUTION CO.
INDUSTRIAL PARK
CITY OF INDUSTRY
CA   92425


 5
HENRY MUSCATEL
819 OAK ST.
NAPA
CA   95303


 6
WIDGET SUPPLY CO.
555 BOGUS AVE.
GERTIE
TENNESSEE   38901

WEITERE ADRESSEN? J ODER N EINEGEN:?N

** PROGRAMM BEENDET **
```



Figur 6 - 5. Flussdiagramm des Programms MAIL. GET #

ORGANISATIONSFORMEN VON BANDDATEIEN

Die Beschreibung von Dateien am Anfang dieses Kapitels gibt das Konzept richtig wieder nach dem im allgemeinen Daten bei der Speicherung strukturiert werden. Dateien werden unterteilt in Aufzeichnungen und Felder. Sie können bei dieser klassischen Form der Datenorganisation bleiben und die hierfür geeignete Programmlogik in CBM-BASIC verwenden, was wir Ihnen empfehlen würden. Aber die tatsächliche Organisation von Dateien auf Bandkassetten hat wenig zu tun mit Aufzeichnungen und Feldern – wie im folgenden deutlich wird.

Jedes numerische Feld muß mit einem Wagenrücklauf-Symbol, CHR$ (13), abgeschlossen werden. Eine aus numerischen Feldern bestehende Datei könnte daher als eine Abfolge von Zahlen angesehen werden, die durch das Wagenrücklauf-Symbol getrennt sind. Das kann wie folgt veranschaulicht werden:

N<CR>N<CR>N<CR>N<CR>N<CR> <CR>= Wagenrücklauf-Symbol

Innerhalb einer numerischen Datei gibt es jedoch nichts, wodurch Felder zu Aufzeichnungen zusammengefaßt werden, oder eine Aufzeichnung sich von der anderen unterscheiden läßt. Es steht völlig im Belieben Ihrer Programmlogik, eine Abfolge von Feldern als eine Reihe von Aufzeichnungen zu begreifen – sofern solche wiederholte Abfolge von Feldern tatsächlich existiert.

Zeichenketten können wahlweise in Aufzeichnungen und Felder unterteilt werden. Zur Trennung von Feldern innerhalb einer Aufzeichnung können Sie Kommas, d.h. CHR$ (44), verwenden, wobei das letzte Feld einer Aufzeichnung mit dem Symbol Wagenrücklauf, CHR$ (13), abgeschlossen wird. Die Speicherung von Zeichenketten auf einer Banddatei in Form von fünf Feldern je Aufzeichnung kann wie folgt veranschaulicht werden:

<CR> Z <,> Z<,> Z<,> Z<,> Z <CR> <,> = Komma-Symbol

Wenn Komma und Wagenrücklauf als Trennsymbole in einer Datei für Zeichenketten benutzt werden, wie eben veranschaulicht, dann müssen alle Felder einer Aufzeichnung mit einer einzigen INPUT#-Anweisung gelesen werden.

Die Verwendung von Komma und Wagenrücklauf als Trennsymbole für Zeichenketten ist jedoch nicht zwingend vorgeschrieben. Beim Arbeiten mit Zeichenketten finden Sie wahrscheinlich bessere Programmlösungen, wenn Sie jedes Feld mit einem Wagenrücklauf als Trennsymbol abschließen. Überlassen Sie wie bei numerischen Feldern der Programmlogik, Felder zu Aufzeichnungen zu gruppieren.

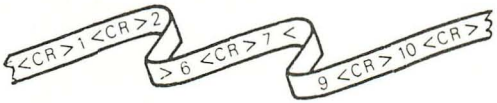
Die erforderliche Programmlogik für den Aufbau von Dateien aus Aufzeichnungen und Feldern ergibt sich gewöhnlich aus der Aufgabenstellung; nehmen Sie als Beispiel die Versandkartei. Ein Programmierer sieht ohne Übung, daß jeder Name mit Adresse eine Aufzeichnung bilden, während Teile des Namens und der Adresse wie einzelne Felder behandelt werden müssen. Es gibt eine Reihe von Verfahren, Teile eines Namens und einer Adresse in Felder zu teilen; möglicherweise ist jedes Verfahren gleich gut. Die Organisation einer Datei wird jedoch von den Erfordernissen Ihres Programms und nicht von der Struktur von Dateien auf CBM-Bandkassetten bestimmt. Schwierigkeiten in der Programmierung werden sich, wenn überhaupt, im Zusammenhang mit der Syntax der Anweisungen PRINT# und INPUT# ergeben.

Wir werden jetzt erlaubte und unerlaubte Syntax bei der Anwendung der Anweisungen PRINT# und INPUT# in einem einfachen Programm kennenlernen.

Geben Sie folgendes Programm über das Tastenfeld ein:

```
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 PRINT#1,I+100
40 NEXT
50 CLOSE 1
60 STOP
70 OPEN 1
80 FOR I=1 TO 10
90 INPUT#1,J
100 PRINT J
110 NEXT
120 CLOSE 1
132 STOP
```

In Zeile 10 wird mit der OPEN-Anweisung die logische Datei #1 auf dem Bandgerät #1 für eine Schreib-Operation geöffnet. Die FOR-NEXT-Schleife in Zeilen 20, 30 und 40 schreibt 10 Zahlen auf die Bandkassette. Jede Zahl wird mit einem Wagenrücklauf-Symbol abgeschlossen, das mit jeder Ausführung der Anweisung PRINT# in Zeile 30 geschrieben wird, ähnlich wie eine PRINT-Anweisung nach jeder Zahlendarstellung auf dem Bildschirm einen Zeilensprung auslöst. In Zeile 50 erfolgt das Schließen der logischen Datei #1. Auf der Bandkassette kann man sich die 10 Zahlen folgendermaßen vorstellen:



CR = RETURN (Wagenrücklauf)

In Zeilen 70 bis 120 erfolgt das Lesen und Darstellen der 10 Zahlen, die in Zeilen 20 bis 50 auf Bandkassette geschrieben wurden.

Lassen Sie das Programm ablaufen und sehen Sie, was sich ereignet.

Nehmen Sie eine leere Bandkassette; bewegen Sie mit dem Finger das Band soweit nach vorne, bis sich seine magnetische Oberfläche in der Schreib-/Leseöffnung befindet, und setzen Sie dann die Bandkassette in das Bandgerät #1 ein. Vergewissern Sie sich, daß keine Taste am Bandgerät niedergedrückt ist.

Lassen Sie sich das Programm mit LIST ausgeben, um sicher zu sein, daß es fehlerfrei in den Arbeitsspeicher eingegeben wurde. Geben Sie jetzt RUN ein. Auf dem Bildschirm erscheint folgende Nachricht:

```
PRESS PLAY AND RECORD ON TAPE #1
```

Der CBM-Computer antwortet mit OK, wenn Sie folgende zwei Tasten am Bandgerät #1 niederdrücken:

```
PRESS PLAY AND RECORD ON TAPE #1
OK
```

Das Magnetband wird aufgespult, während die 10 Zahlen 101 bis 110 auf Bandkassette geschrieben werden. Nach dem Einschreiben dieser 10 Zahlen hält der Bandantrieb an, und folgende Nachricht erscheint auf dem Bildschirm:

```
BREAK IN 60
READY
```

Unterhalb der Nachricht blinkt der Kursor. Der Programmabbruch erfolgte in Zeile 60 durch die Anweisung STOP. Wenn Sie jetzt die STOP-Taste am Bandgerät #1 drücken, springen die Tasten PLAY und RECORD zurück. Drücken Sie die REWIND-Taste nieder, um das Magnetband völlig zurückzuspulen, und dann drücken Sie wiederum die STOP-Taste, um die REWIND-Taste zurückspringen zu lassen. Lassen Sie jetzt die zweite Programmhälfte ablaufen, indem Sie folgendes eingeben:

```
GOTO 70
```

Auf dem Bildschirm erscheint die Nachricht PRESS PLAY ON TAPE #1. Drücken Sie die PLAY-Taste am Bandgerät #1. Daraufhin antwortet der Computer mit OK:

```
PRESS PLAY ON TAPE 1
OK
```

Eine Zeitlang geschieht nichts; der Bandantrieb spult das Magnetband auf, bis die 10 vorher eingeschriebenen Zahlen erkannt worden sind. Danach werden diese 10 Zahlen spaltenweise auf dem Bildschirm dargestellt:

```
101
102
103
104
105
106
107
108
109
110
BREAK IN 130
READY
```

Die 10 Zahlen werden in einer Spalte dargestellt, da jede Zahlendarstellung durch erneute Ausführung der Anweisung PRINT in Zeile 100 erfolgt, die jedesmal mit einem Zeilensprung verbunden ist.

Die letzte Nachricht wird durch Ausführung der Anweisung STOP in Zeile 130 verursacht.

```
BREAK IN 130
READY
```

Wenn Sie vergessen, das Band zurückzuspulen, ehe Sie GOTO 70 eingeben, dann wird das Magnetband bis zum Ende nach Daten abgesucht, die nur am Anfang des Magnetbands zu finden sind. Sie müssen das Bandgerät und den Programmablauf stoppen. Spulen Sie die Bandkassette zurück, aber schließen Sie vor einem erneuten Programmstart die Datei mit der logischen Nummer 1 durch direktes Eingeben der Anweisung:

```
CLOSE 1
```

Das Programm wird erneut gestartet mit:

```
GOTO 70
```

Listen Sie das Programm wieder auf und schließen Sie die Anweisung PRINT in Zeile 100 mit einem Semikolon ab:

```
100 PRINT J;
```

Spulen Sie das Magnetband zurück und geben Sie dann den Befehl GOTO 70 ein.

Auf dem Bildschirm erscheint erneut die Nachricht PRESS PLAY ON TAPE # 1. Nach Drücken der Taste PLAY erscheint die Antwort OK auf dem Bildschirm. Nach kurzer Pause werden die 10 vom Band gelesenen Zahlen auf einer Zeile des Bildschirms dargestellt:

```
101  102  103  104  105  106  107  108  109  110
BREAK IN 130
READY
```

Als weiteren Versuch werden wir die Anweisungen in Zeile 80 bis 110 derart ändern, daß zur Eingabe der 10 Nummern nur eine INPUT-Anweisung erforderlich ist:

```
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 PRINT#1,I+100
40 NEXT
50 CLOSE 1
60 STOP
70 OPEN 1
80 INPUT#1,N(1),N(2),N(3),N(4),N(5),N(6),N(7),N(8),N(9),N(10)
90 FOR I=1 TO 10
100 PRINT N(I);
110 NEXT
120 CLOSE 1
130 STOP
```

Spulen Sie wiederum das Magnetband zurück und lassen Sie den zweiten Teil des Programms dadurch ablaufen, daß Sie GOTO 70 am Tastenfeld eingeben.

Sie werden wiederum die Aufforderung PRESS PLAY ON TAPE #1 erhalten, nach deren Ausführung die 10 Zahlen von der Bandkassette gelesen und in einer einzigen Zeile auf dem Bildschirm dargestellt werden, wie im vorangegangenen Beispiel. Demnach ergibt sich kein Unterschied beim Lesen der 10 Zahlen von Bandkassette, wenn die INPUT#-Anweisung mit 10 Variablen in der Parameter-Liste einmal ausgeführt wird, oder wenn die INPUT#-Anweisung mit 1 Variablen zehnmal ausgeführt wird.

Wir machen weitere Versuche mit den Trennsymbolen zwischen Feldern und ändern hierzu den ersten Teil des Programms, der Daten auf Bandkassette schreibt, wie folgt:

```
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 PRINT#1,I+100
40 NEXT
45 C$=CHR$(59)
46 PRINT#1,M(1);C$;M(2);C$;M(3);C$;M(4);C$;M(5)
47 PRINT#1,M(6);C$;M(7);C$;M(8);C$;M(9);C$;M(10)
50 CLOSE 1
60 STOP
70 OPEN 1
80 FOR I=1 TO 10
90 INPUT#1,J
100 PRINT J
110 NEXT
120 CLOSE 1
130 STOP
```

In Zeile 45 wird als Trennsymbol ein Semikolon, CHR$ (59), eingeführt. Spulen Sie das Magnetband zurück, bringen Sie dann den Anfang seiner magnetischen Oberfläche in das Schreib-Lesefenster und setzen es in das Bandgerät ein. Mit allen Tasten am Bandgerät in der oberen Stellung geben Sie jetzt RUN ein. Führen Sie dann die Anweisungen auf dem Bildschirm durch Drücken der Tasten PLAY und RECORD aus. Die Daten werden erfolgreich auf Band gespeichert, und folgende Nachricht erscheint:

```
BREAK IN 60
READY
```

Spulen Sie die Kassette zurück und geben Sie GOTO 70 ein.

Bei Erscheinen der Anweisung drücken Sie die Taste PLAY am Bandgerät #1. Jedoch: die Daten werden nicht erfolgreich zurückgelesen und auf dem Bildschirm erscheint eine Fehlermeldung:

```
FILE DATA ERROR IN 90
READY
```

Die Schlußfolgerung ist: zur Trennung numerischer Felder können Sie keine andere Zeichensetzung als das Wagenrücklauf-Symbol verwenden. Für Felder mit Zeichenketten sind die Symbole Komma und Wagenrücklauf zulässig. Ändern Sie zur Bestätigung das Programm wie folgt:

```
5 DATA EINS,ZWEI,DREI,VIER,FUENF,SECHS,SIEBEN,ACHT,NEUN,ZEHN
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 READ M$(I)
40 NEXT
45 C$=CHR$(44)
46 PRINT#1,M$(1);C$;M$(2);C$;M$(3);C$;M$(4);C$;M$(5)
47 PRINT#1,M$(6);C$;M$(7);C$;M$(8);C$;M$(9);C$;M$(10)
50 CLOSE 1
60 STOP
70 OPEN 1
80 FOR I=1 TO 10
90 INPUT#1,J$
100 PRINT J$
110 NEXT
120 CLOSE 1
130 STOP
```

Spulen Sie die Bandkassette zurück, lassen Sie das Magnetband bis zum Erscheinen seiner magnetischen Oberfläche vorwärts laufen, setzen Sie es in das Bandgerät #1 ein und geben Sie RUN ein. Nach Anweisung auf dem Bildschirm drücken Sie die Tasten PLAY und RECORD am Bandgerät #1. Die Daten werden ohne Fehlermeldung auf Bandkassette gespeichert. Sobald die Nachricht:

```
BREAK IN 60
READY
```

erscheint, spulen Sie das Magnetband wieder zurück und geben am Tastenfeld GOTO 70 ein.

Nach Aufforderung drücken Sie die Taste PLAY. Sie werden dann die Zeichenketten EINS und SECHS auf dem Bildschirm sehen, gefolgt von der Fehlermeldung:

```
STRING TOO LONG ERROR IN 90
READY
```

Was lief falsch? Das Problem entstand mit der INPUT#-Anweisung in Zeile 90. Eine INPUT#-Anweisung liest alle Zeichenketten-Felder, die sie bis zum ersten Wagenrücklauf-Symbol vorfindet. Bei der ersten Ausführung der INPUT#-Anweisung in Zeile 90 werden daher die Variablen M$ (1) bis M$ (5) eingelesen; da diese Variablen durch Kommas getrennt sind, werden sie als zusammenhängende Felder und nicht als einzelne Aufzeichnungen verstanden, so daß nur M$ (1) gelesen und J$ zugewiesen wird. Bei der zweiten Ausführung der Anweisung INPUT# in Zeile 90 werden die Variablen M$ (6) bis M$ (10) gelesen, da sie zwischen zwei Wagenrücklauf-Symbolen liegen. Wiederum erfährt nur eine Variable, M$ (6), eine Zuordnung zu J$. Bei der dritten Ausführung der INPUT#-Anweisung in Zeile 90 sind keine Daten zum Lesen übrig und eine Fehlermeldung wird ausgegeben. Hierdurch erklären sich die Angaben auf dem Bildschirm. Um den Fehler zu beheben, müssen wir die INPUT#-Anweisungen mit der gleichen Anzahl von Variablen in der Parameter-Liste versehen, wie wir es in der PRINT#-Anweisung taten. Sehen Sie sich folgendes Programm an:

```
5 DATA EINS,ZWEI,DREI,VIER,FUENF,SECHS,SIEBEN,ACHT,NEUN,ZEHN
10 OPEN 1,1,1
20 FOR I=1 TO 10
30 READ M$(I)
40 NEXT
45 C$=CHR$(44)
46 PRINT#1,M$(1);C$;M$(2);C$;M$(3);C$;M$(4);C$;M$(5)
47 PRINT#1,M$(6);C$;M$(7);C$;M$(8);C$;M$(9);C$;M$(10)
50 CLOSE 1
60 STOP
70 OPEN 1
80 INPUT#1,N$(1),N$(2),N$(3),N$(4),N$(5)
90 INPUT#1,N$(6),N$(7),N$(8),N$(9),N$(10)
100 FOR I=1 TO 10
105 PRINT N$(I);" ";
110 NEXT I
120 CLOSE 1
130 STOP
```

Wenn Sie wiederum die Programmhälften nacheinander ablaufen lassen, dann wird Ihnen die zweite Programmhälfte folgende Bildschirmdarstellung liefern:

```
EINS ZWEI DREI VIER FUENF SECHS SIEBEN ACHT NEUN ZEHN
BREAK IN 130
READY
```

Es gibt noch einige weitere Versuche, deren Ausführung sich für Sie lohnt:

Kann eine einzige INPUT#-Anweisung eine Abfolge von Zeichenketten lesen, die jeweils durch ein Wagenrücklauf-Symbol getrennt sind? Ändern Sie für diesen Versuch Zeile 45 in der letzten Programmversion derart, daß C$ der Wert CHR$ (13) zugewiesen wird. Dann wiederholen Sie den Programmlauf.

Wie steht es mit gemischten Feldern aus Zahlen und Zeichenketten innerhalb einer einzigen Datei? Um das zu erfahren, erzeugen Sie die 10 Zeichenketten-Variablen M$ (I) wie in der letzten Programmversion und erzeugen noch zusätzlich die numerischen Variablen M (I) durch Hinzufügen der folgenden Anweisung in Zeile 35:

```
35 M(I)=I+100
```

Bilden Sie jetzt in der PRINT#-Anweisung in Zeile 46 eine Parameter-Liste mit beiden Variablen-Typen, M$ (I) und M (I). Sehen Sie dann zu, welche INPUT#-Anweisungen in Zeilen 80 und 90 erforderlich sind, um diese Folge von Variablen richtig zu lesen.

ARBEITEN MIT DATEIEN AUF DISKETTEN

Auf Disketten können sowohl Dateien für Programme wie Dateien für Daten angelegt werden. Programm-Dateien dienen der Speicherung von BASIC-Programmen. Dateien für Daten dienen der Speicherung von Zahlen und Zeichenketten.

Man unterscheidet drei Typen von Dateien auf Disketten:

1. **Sequentielle Dateien,** mit denen Daten sehr kompakt gespeichert werden können, die aber beschränkte Zugriffsmöglichkeiten zu den Dateien bieten.

2. **Relative Dateien,** die eine größere Oberfläche der Diskette benötigen, um die gleiche Datenmenge wie eine Sequentielle Datei zu speichern, bei denen aber Dateizugriff und Änderung der Dateiorganisation leichter auszuführen sind.
3. **Wahlfrei Ansprechbare Dateien,** deren Struktur von Ihrer Programmlogik abhängt.

Sequentielle und Relative Dateien werden in diesem Kapitel beschrieben. Die Beschreibung Wahlfrei Ansprechbarer Dateien erfolgt in Kapitel 7.

VERGLEICH VON DATEIEN AUF DISKETTEN UND BANDKASSETTEN

Das Arbeiten mit Disketten-Dateien unterscheidet sich vom Arbeiten mit Band-Dateien in folgenden Punkten:

1. Der Zugriff auf Disketten-Dateien ist sehr schnell im Vergleich zum Zugriff auf Band-Dateien.
2. Eine Diskette kennt kein "Anfang" oder "Ende" der Speicherfläche, wie eine Bandkassette. Auf jeden Punkt der Disketten-Oberfläche kann mit gleicher Schnelligkeit zugegriffen werden, im Gegensatz zum Magnetband.

Das Arbeiten mit Dateien auf Kassette und Diskette unterscheidet sich auffällig durch das völlig verschiedene Format der Datenspeicherung und die Verfahren des Dateizugriffs. Die mechanische Geschwindigkeit der beiden Speichermedien hat hiermit wenig zu tun; die Rotationsgeschwindigkeit der Diskette ist vergleichbar mit der Laufgeschwindigkeit der Bandkassette.

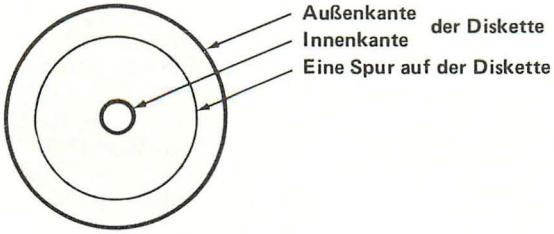
Die Datenspeicherung auf Bandkassette erfolgt auf einer von Bandanfang bis Bandende durchlaufenden Spur; der Bandantrieb bewegt das Magnetband an feststehenden Schreib-/Leseköpfen vorbei, um Zugriff auf einen Teil des Bandes zu ermöglichen. Das Speichern von Daten auf Disketten erfolgt im Gegensatz dazu auf einer großen Zahl konzentrischer, kreisförmiger Spuren Schreib- und Leseköpfe befinden sich an einem beweglichen Arm, der auf jede Spur der Diskette positioniert werden kann. Die Rotation der Diskette ist erforderlich, um den gewünschten Sektor der gewählten Spur unter den Lese- oder Schreibkopf zu bringen.

Zur Benutzung von Disketten müssen Sie nicht die Technik der Datenspeicherung auf Disketten kennen, einige Kenntnisse werden aber hilfreich sein, um Programme für Disketten-Dateien wirkungsvoller zu gestalten. Wir werden deshalb mit der Beschreibung der Art und Weise der Datenspeicherung auf Disketten beginnen.

DATENSPEICHERUNG AUF DISKETTEN

Disketten benutzen eine Anzahl konzentrischer Spuren zur Datenspeicherung. Die Spuren sind wiederum unterteilt in Sektoren.

Um eine Vorstellung von einer einzelnen Spur zu haben, zeichnen wir einen Kreis zur Darstellung der Diskette und einen kleineren konzentrischen Kreis zur Darstellung einer Spur auf der Diskette. Das ergibt folgendes Bild:



Die Zahl der Spuren auf einer Diskette sind je nach Floppy Disk-Gerät verschieden. Manche Floppy Disk-Geräte beschreiben beide Seiten der Diskette; andere beschreiben nur eine Seite der Diskette. Die Floppy Disk-Speicher CBM 3040, 4040 und 8050 beschreiben eine Seite der Diskette; die Zusammenstellung in Tabelle 6-3 zeigt, daß die Speichereinheiten 3040 und 4040 auf der Diskette 35 Spuren beschreiben, die Speichereinheit 8050 dagegen 77 Spuren.

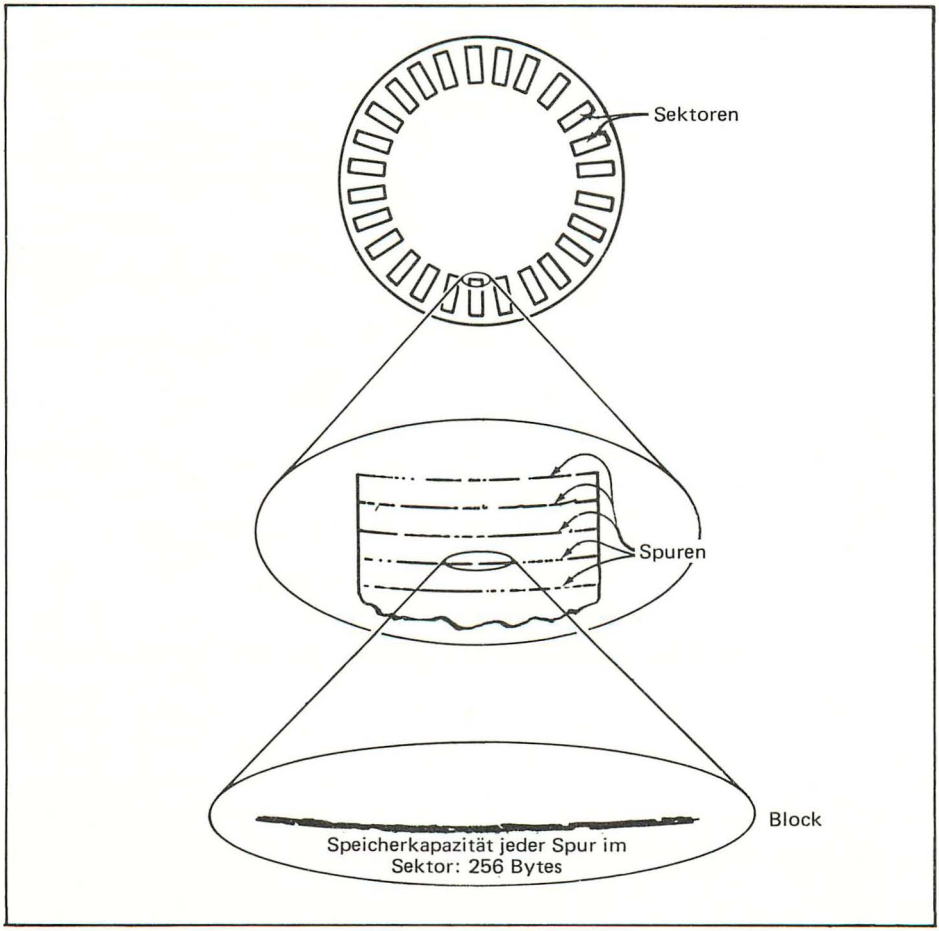


Bild 6-6. Speicheroberfläche einer Diskette

Daten werden nicht über die ganze Länge einer Spur auf der Diskette gespeichert. Die Adressierung von Dateien auf der Diskette würde hierdurch sehr erschwert werden, da keine zwei Spuren die gleiche Länge haben, und daher auch nicht die gleichen Datenmenge speichern können.
Zur Lösung dieses Problems werden Spuren in Sektoren unterteilt. Jeder Sektor kann die gleiche Datenmenge speichern. Bei den Speichern 3040, 4040 und 8050 werden in jedem Sektor 256 Symbole (Bytes) gespeichert. Diese Anordnung der Daten ist in Figur 6-6 dargestellt. Eine Datenmenge von 256 Byte heißt Block (Figur 6-6).

Die meisten Floppy Disk-Speicher haben auf jeder Spur die gleiche Anzahl von Sektoren, auch wenn die Spur am Rand der Diskette sehr viel länger ist als nahe dem Zentrum. Die CBM-Geräte 3040, 4040 und 8050 nutzen die größere Länge der Spuren am äußeren Rand der Diskette, indem sie auf die längeren Spuren mehr Sektoren legen. Tabelle 6-3 gibt die Zahl der Sektoren auf den verschiedenen Spuren an. Die Zählung der Spuren beginnt mit 0 für die äußerste Spur. Die innerste Spur trägt die höchste Spurnummer.

Wenn Sie mit der Hand eine CBM-Diskette in ihrer Schutzhülle drehen, dann können Sie in dem kleinen kreisförmigen Fenster nahe der Mitte der Schutzhülle ein kreisförmiges Loch in der Diskette feststellen. Bei Disketten mit einem einzelnen Loch ist die Anordnung der Sektoren auf der Diskette noch nicht festgelegt (soft-sektoriert). Im Gegensatz dazu gibt es Disketten mit Sektoren fester Anordnung. Derartige Disketten weisen soviele Löcher auf, wie es Sektoren auf ihrer Oberfläche gibt (hard-sektoriert). CBM-Disketten sind soft-sektoriert.

Tabelle 6-3. CBM Floppy Disk-Speicher: Speicherkapazität und -organisation

Merkmal	CBM 3040/4040 mit 2 Laufwerken für 2 Disketten		CBM 8050 mit 2 Laufwerken für 2 Disketten	
Maximale Speicherkapazität	176,640 Bytes/Diskette		533,248 Bytes/Diskette	
Verfügbare Kapazität —Serielle Dateien —	170,180 Bytes/Diskette		521,208 Bytes/Diskette	
Verfügbare Kapazität — Relative Dateien —	nicht verfügbar		bis 517398 Bytes/Diskette	
Anzahl der Spuren	35		77	
Anzahl der Sektoren	Spur #	Sektoren	Spur #	Sektoren
	1-17 18-24 25-30 31-35	21 19 18 17	1-39 40-53 54-64 65-77	29 29 25 23
Bytes je Block	256		256	
Anzahl der Blöcke	690		2083	
Block-Verfügbarkeits-Tabelle:	Spur #17 2 Sektoren		Spur #38 2 Sektoren	
Inhaltsverzeichnis:	Spur #17 18 Sektoren		Spur #39 29 Sektoren	

VERZEICHNIS DES DISKETTEN-INHALTS

Auf jeder Diskette dienen zwei Spuren der Identifizierung und Inhaltsangabe einer Diskette.

Das Inhaltsverzeichnis einer Diskette enthält den Namen der Diskette, die Namen aller Dateien und die Adressen der Sektoren, mit denen die Dateien beginnen.

Die Sektor-Belegungs-Liste (BAM) gibt den Belegungszustand der Sektoren auf der Diskette an, d.h. welche Sektoren frei sind und welche Sektoren von Dateien belegt sind. Die Benutzung dieser Liste für Dateien mit wahlfreiem Zugriff wird in Kapitel 7 beschrieben.

Für Dateien auf Bandkassette ist kein Inhaltsverzeichnis am Bandanfang erforderlich. Wenn beispielsweise 10 Dateien auf Bandkassette gespeichert wurden und auf die sechste Datei zugegriffen werden muß, dann kann ein Inhaltsverzeichnis der Dateien am Bandanfang die Suchzeit für das Auffinden der sechsten Datei nicht verkürzen. Da eine Banddatei jede Länge haben kann, gibt es keine Möglichkeit, aus der Nummer einer Banddatei auf ihre Lage auf dem Magnetband zu schließen. Sie können Ihr Glück versuchen und das Magnetband soweit aufspulen, daß es kurz vor der gewünschten Datei steht und sich hierdurch die Suchzeit verringert. Im allgemeinen muß das Bandgerät die ersten Dateien überlesen, bis es den Anfang der gewünschten Datei gefunden hat.

Im Gegensatz dazu kann ein Floppy Disk-Speicher direkt den Datei-Anfang auf der Oberfläche der Diskette ansteuern, da jeder Sektor auf der Diskette gleichermaßen zugänglich ist. Hierzu verfügt jede Diskette über ein Inhaltsverzeichnis der Dateinamen und der Sektoradressen der Dateianfänge aller auf der Diskette gespeicherten Dateien. Das Inhaltsverzeichnis gibt auch den Typ der Datei und die augenblickliche Länge der Datei an. Wird eine Disketten-Datei geöffnet, dann liest der Floppy Disk-Speicher zunächst das Inhaltsverzeichnis auf der Diskette, um die Sektoradresse der zu öffnenden Datei zu finden. Der Lese-/Schreibarm des Floppy Disk-Speichers kann dann direkt auf die Anfangsspur der zu öffnenden Datei gesetzt werden.

Wie aber sehen Aufzeichnungen in einer Disketten-Datei aus?

RELATIVE DATEIEN

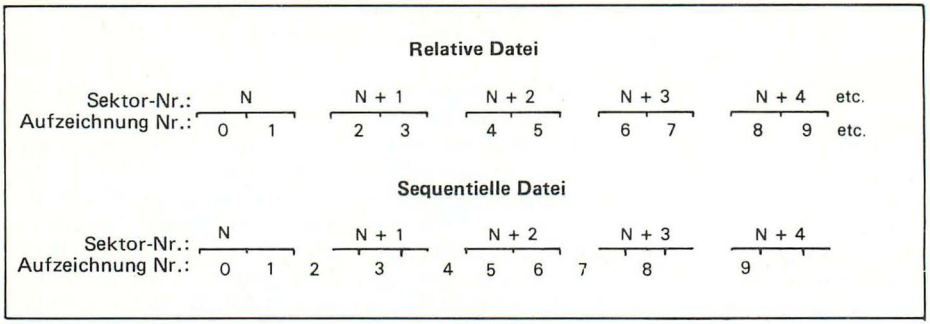
In einer Relativen Datei haben alle Aufzeichnungen die gleiche Länge. Die Sektoradresse jeder einzelnen Aufzeichnung ist daher im Falle relativer Dateien leicht zu berechnen. Ein Beispiel: Nehmen Sie an, zwei Aufzeichnungen belegen genau einen Sektor. Dieser Fall wird kaum zufällig eintreten, aber er ist einfach für unsere Veranschaulichung. Die zehnte Aufzeichnung in dieser relativen Datei kann daher im fünften Sektor gefunden werden. Relative Dateien sind in CBM BASIC ab Version 4.0 mit dem Disketten-Betriebssystem DOS ab Version 2.0 möglich.

SEQUENTIELLE DATEN

Die Aufzeichnungen einer sequentiellen Datei können unterschiedliche Länge haben. Eine einzelne Aufzeichnung in einer sequentiellen Datei kann daher nicht mehr durch Berechnung des Sektors gefunden werden, da die Länge der einzelnen Aufzeichnungen unbekannt ist. Der Floppy Disk-Speicher kann zwar direkt an den Anfang einer Sequentiellen Datei gehen, da die Sektoradresse des Dateianfangs aus dem Inhaltsverzeichnis der Diskette zu ersehen ist, die einzelnen Aufzeichnungen

innerhalb der Sequentiellen Datei können jedoch nur wie bei einer Bandkassette sequentiell gelesen werden. Ein Beispiel: es gibt keine Möglichkeit, zur zehnten Aufzeichnung einer Sequentiellen Datei zu gelangen, ohne zuvor die Aufzeichnungen 1 bis 9 gelesen zu haben. Figur 6-7 vergleicht die Sektorbelegung durch 10 Aufzeichnungen in einer Relativen und in einer Sequentiellen Datei.

Alle Versionen von CBM BASIC ermöglichen Sequentielle Dateien.



Figur 6-7 Zählung von Sektor und Aufzeichnung bei Relativen und Sequentiellen Dateien

VERGLEICH VON RELATIVEN UND SEQUENTIELLEN DATEIEN

Wenn Aufzeichnungen sequentiell gelesen werden müssen, dann geht die Eigenschaft einer Diskette verloren, willkürlichen Zugriff auf einzelne Aufzeichnungen zu ermöglichen. Warum sich dann mit Sequentiellen Dateien abgeben? Die Antwort ist, daß Sequentielle Dateien eine dichtere Informationsspeicherung erlauben als Relative Dateien. Sequentielle Dateien nutzen die Oberfläche der Diskette besser aus als Relative Dateien. Betrachten Sie zur Veranschaulichung die folgenden zwei Namen und Adressen:

Cornelius J. Winkleberger
257631 Avenue of the Americas
Billinghampton California 92804

Joe R. Smith
5 N St.
York Iowa 50307

Nehmen wir an, diese beiden Namen und Adressen sind Auszug aus einer Versandliste. Jeder Name und jede Adresse werden in Form von Aufzeichnungen gespeichert. Eine Relative Datei muß jeder Aufzeichnung von Name und Adresse den gleichen Platz auf der Diskette zuweisen. Um Abkürzungen zu vermeiden, muß der zugewiesene Platz auf der Diskette ausreichen, auch den längsten Namen und die längste Adresse aufzunehmen. Bei allen kürzeren Namen und Adressen bleibt Platz auf der Diskette unbenutzt; und das bedeutet vergeudeter Speicherplatz.

Eine Sequentielle Datei weist dagegen jedem Namen und jeder Adresse den Speicherplatz zu, den sie benötigen, wie kurz oder wie lang auch der einzelne Name oder die einzelne Adresse sind. Kein Platz auf der Diskette bleibt ungenutzt, und es wird daher kein Speicherplatz vergeudet.

Relative Dateien erlauben ohne Beschränkung Zugriff oder Änderung. Da Relative Dateien aus Aufzeichnungen fester Länge bestehen, die einzeln adressiert werden können, können Sie auf einzelne Aufzeichnungen zugreifen, um sie zu lesen oder zu ändern. Ein Beispiel: In einer Relativen Datei mit 20 Namen und Adressen können Sie den zehnten Namen ändern, während die restlichen Aufzeichnungen unverändert bleiben. Sie können eine Relative Datei solange um Aufzeichnungen erweitern, wie auf der Diskette Speicherplatz verfügbar ist. Sie können auch einzelne Aufzeichnungen in einer Relativen Datei löschen.

Sequentielle Dateien müssen Sie dagegen ähnlich wie Dateien auf Bandkassette behandeln. Aufzeichnungen müssen sequentiell gelesen werden, beginnend mit der ersten Aufzeichnung der Datei. Sie können neue Aufzeichnungen an das Ende der Sequentiellen Datei anfügen, aber nicht neue Aufzeichnungen in eine bestehende Sequentielle Datei einfügen. Im letzten Fall müssen Sie die gesamte Sequentielle Datei noch einmal schreiben, wobei während des Überspielens Gelegenheit für Änderungen an den Aufzeichnungen besteht. Das Ergebnis des Vergleichs ist, daß Sequentielle Dateien die Disketten-Oberfläche besser ausnutzen, aber schwieriger zu handhaben sind.

ADRESSIERUNG VON SEKTOREN

Es ist unwahrscheinlich, daß die Sektoren einer Datei auf der Oberfläche der Diskette physikalisch aufeinander folgen. Ein Beispiel: Wenn Sie Aufzeichnungen zu einer bestehenden Datei hinzufügen, können die neuen Aufzeichnungen in den Anfang der folgenden Datei hineinreichen; die Datei muß dort fortgesetzt werden, wo noch unbeschriebene Sektoren auf der Oberfläche der Diskette verfügbar sind. Beim Löschen von Aufzeichnungen schrumpft die Datei. Leer gewordene Sektoren müssen anderen Dateien zugänglich gemacht werden. Die Betriebslogik eines Floppy Disk-Speichers nimmt daher an, daß die Sektoren einer Datei über die Oberfläche der Diskette verstreut sind. Für das Arbeiten mit Sequentiellen Dateien entsteht hieraus kein Problem. Solange jeder Sektor einen Hinweis auf seinen Nachfolger enthält, kann der Floppy Disk-Antrieb seinen Weg über die Diskette finden, wobei er die Sequentielle Datei Sektor für Sektor liest. Das Problem ist verwickelter bei Relativen Dateien, da die Floppy Disk-Logik in der Lage sein muß, die Adressen der einzelnen Aufzeichnungen zu berechnen. Der Abstand einer Aufzeichnung vom Datei-Anfang muß in die Zahl der dazwischenliegenden Sektoren umgerechnet werden. Nehmen wir wieder das Beispiel einer Datei mit zwei Aufzeichnungen je Sektor: Die Aufforderung zum Zugriff auf die zehnte Aufzeichnung entspricht der Aufforderung, auf den fünften Sektor der Relativen Datei zuzugreifen. Da Sektoren jedoch über die Oberfläche der Diskette verstreut sind, muß für die Relative Datei ein Lageverzeichnis der Sektoren geführt werden. Dieser Gedanke kann wie folgt veranschaulicht werden:

Aufzeichnung Nr.	Beginn der Aufzeichnung in Sequentieller Sektor-Nr.	Tatsächliche Spur- und Sektoradressen	
		Spur-Nr.	Sektor-Nr.
1	1	11	4
2	1		
3	2	11	5
4	2		
5	3	11	6
6	3		
7	4	13	9
8	4		
9	5	13	10
10	5		
11	6	9	3

Aufzeichnung Nr. 6 beispielsweise liegt im dritten Sektor der Datei. Die physikalische Lage dieses Sektors ist durch Sektor 6 auf Spur 11 gegeben.

Für die Speicherung der Information über die Lage der Sektoren einer Datei ist eine bestimmte Zahl von Sektoren reserviert. Die Existenz dieser reservierten Sektoren bewirkt, daß beim Floppy Disk-Speicher CBM 8050 nicht die gesamte Speicherkapazität einer Diskette für Relative Dateien ausgenutzt werden kann (Tabelle 6-3).

BETRIEBSPROGRAMME FÜR DISKETTE-DATEIEN

Sequentielle und Relative Dateien erfordern unterschiedliche Programmlogiken. Eine Reihe weiterer Programmanweisungen ermöglicht die Verwaltung von Dateien auf Disketten.

DATEINAMEN

Namen für Disketten-Dateien folgen den normalen Bezeichnungsregeln in CBM BASIC. In der Regel sind Dateinamen auf 16 Symbole beschränkt und es ist ein guter Grundsatz, sich an diese Regel selbst dort zu halten, wo diese Begrenzung nicht verlangt wird.

Das Betriebssystem für Disketten (DOS) benutzt Anweisungen, in denen Dateien durch ihren Namen angesprochen werden. Sie können den vollständigen Dateinamen angeben oder einige der ersten Symbole gefolgt von dem Sternsymbol, was bewirkt, daß der erste Dateiname mit übereinstimmenden Anfangssymbolen ausgesucht wird. Hier einige Beispiele:

Eingabe: PAR*
angesprochene Dateien: PARITY
PARITY,SEC
PARITY,N12
PARTITION
etc.

Die erste Datei mit den Anfangsbuchstaben PAR wird angesprochen.

Sie können auch dadurch nach einer Datei suchen, daß Sie nur einige Symbole aus dem Dateinamen angeben. Für nicht angegebene Symbole müssen Fragezeichen (?) eingesetzt werden. Hier ein Beispiel:

Eingabe: N??,SEQ
angesprochene Dateien: NUM,SEQ
NXY,SEQ
NAB,SEQ
NRA,SEQ
etc.

Alle Dateien mit den Anfangssymbolen N??, SEQ in ihrem Namen, wobei ? für beliebige Symbole steht.
Die erste derartige Datei wird angesprochen.

Auch die Verbindung von Fragezeichen und Sternsymbol ist möglich.

Eingabe: NUM??*
angesprochene Dateien:

Alle Dateien mit 5 oder mehr Symbolen in ihrem Namen, wovon die ersten 3 Symbole NUM sind.
Die erste derartige Datei wird angesprochen.

VERSIONEN DES BETRIEBSSYSTEMS FÜR DISKETTEN

Anweisungen zum Arbeiten mit Diskette-Dateien in CBM BASIC beruhen auf einer Anzahl von Programmen, die zusammenfassend Betriebssystem für Disketten (DOS = DISK-OPERATING-SYSTEM) bezeichnet werden. Sie müssen nur wenig über ein Betriebssystem für Disketten wissen, um es benutzen zu können, wie Sie auch nur wenig über einen BASIC-Übersetzer wissen müssen, um Programme in BASIC schreiben zu können. Sie sollten jedoch beachten, daß einige Versionen des CBM-Betriebssystems für Disketten vorliegen. Jede Version wird durch eine Zahl nach der Angabe DOS gekennzeichnet. Gegenwärtig sind die Versionen DOS 1.x bis DOS 2.x in Verwendung (x kann eine der Zahlen 0-9 sein).

VERSIONEN VON CBM BASIC

Erinnern Sie sich, daß vier Versionen von CBM BASIC in allgemeinem Gebrauch sind. BASIC 3.0 und frühere Versionen wurden mit allen CBM-Computern bis März 1980 ausgeliefert. Seit dieser Zeit wird BASIC 4.0 ausgeliefert.

Die Versionen BASIC 1.0, 2.0 und 3.0 sind sehr ähnlich. Wie im Vorwort erwähnt, beziehen wir uns auf diese drei Versionen von BASIC mit der gemeinsamen Angabe BASIC<3.0. Die Version 4.0 wird als BASIC 4.0 angegeben.

BASIC<3.0 ermöglicht Sequentielle Dateien und Dateien mit wahlfreiem Zugriff. BASIC 4.0 ermöglicht Sequentielle und Relative Dateien sowie Dateien mit wahlfreiem Zugriff.

BASIC 4.0 erlaubt die Verwendung aller Anweisungen der Versionen 1.0 bis 3.0. Wenn Sie einen CBM-Computer für BASIC 4.0 besitzen, können Sie jede Anweisung für das Arbeiten mit Disketten verwenden. Das umgekehrte ist nicht möglich. Wenn Ihr CBM-Computer beispielsweise mit BASIC 2.0 arbeitet, dann können Sie nicht jede Anweisung aus BASIC 4.0 verwenden.

BASIC 4.0 ermöglicht nicht den Betrieb eines zweiten Bandgeräts, wenn mit Floppy Disk-Speichern gearbeitet wird.

BASIC 4.0 nimmt bei fehlender Geräteadresse den Anschluß eines Floppy Disk-Speichers an; bei fehlender Angabe einer Geräteadresse wird von BASIC 4.0 die Adresse 8 angenommen. Im Gegensatz dazu nehmen die Versionen 1.0 bis 3.0 die Bandkassette mit der Geräteadresse 1 an, falls die Angabe einer Gerätenummer fehlt. Obwohl BASIC 4.0 alle Anweisungen der Versionen 1.0 bis 3.0 ausführt, gibt es einige Fehlermeldungen und Unverträglichkeiten der Statusmeldungen, wenn Dateianweisungen aus BASIC<3.0 in BASIC 4.0 verwendet werden. Beispielsweise ermöglicht BASIC<3.0 nicht das Anlegen Relativer Dateien; wenn Sie eine Datei in BASIC<3.0 öffnen und nicht den Dateityp angeben, wird auf einem Computer mit BASIC 4.0 eine Relative Datei geöffnet. Wenn Sie andererseits eine Dateioperation mit Anweisungen aus BASIC<3.0 durchführen und diese Operation in BASIC<3.0 regelwidrig ist, wohl aber den Regeln in BASIC 4.0 entspricht, dann wird die Fehleranzeige am Floppy Disk-Speicher rot aufleuchten, die Operation mit der Diskette wird nicht ausgeführt, aber BASIC 4.0 meldet einen fehlerfreien Status der Floppy Disk-Funktion.

ÖFFNEN EINER DISKETTEN-DATEI

Jeder Floppy Disk-Speicher verfügt über 12 Pufferspeicher zum Zugriff auf Dateien, die in den Laufwerken 0 und/oder 1 gespeichert sind. Sobald auf irgendeine Diskettendatei zugegriffen wird, werden zwei der Pufferspeicher für Verwaltungstätigkeiten benutzt. Der eigentliche Zugriff auf die Dateien erfolgt über die verbleibenden 10 Pufferspeicher.

Für jede geöffnete Sequentielle Datei sind zwei Pufferspeicher erforderlich. Jede geöffnete Relative Datei belegt drei Pufferspeicher. In BASIC<3.0 können daher bis zu 5 Sequentielle Dateien gleichzeitig geöffnet sein. Die Zahl gleichzeitig geöffneter Dateien in BASIC 4.0 hängt von ihrer Zusammensetzung aus Sequentiellen Dateien und Dateien mit wahlfreiem Zugriff ab. Folgende Kombinationen sind zulässig:

0 Relative und 5 Sequentielle Dateien
1 Relative und 3 Sequentielle Dateien
2 Relative und 2 Sequentielle Dateien
3 Relative und 0 Sequentielle Dateien

SEKUNDÄRE ADRESSEN IN BASIC<3.0

In BASIC<3.0 werden die 16 Sekundären Adressen 0 bis 15 verwendet. Jede Dateieröffnung in BASIC<3.0 mit der Anweisung OPEN muß eine Sekundäre Adresse enthalten. In BASIC 4.0 werden Sekundäre Adressen automatisch eingegeben.

Folgende Sekundäre Adressen werden in BASIC<3.0 verwendet:

0 Die Adresse 0 wird verwendet, um Programme von der Diskette in den Arbeitsspeicher des CBM-Computers zu laden.

1 Die Adresse 1 wird verwendet, um Programme im Arbeitsspeicher des Computers in eine Disketten-Datei zu übertragen.

2-14 Die Adressen 2-14 werden für den Zugriff auf Dateien benutzt. Jede dieser Sekundären Adressen kann verwendet werden, sofern sie nicht schon zum Öffnen einer anderen Datei benutzt worden ist.

15 Die Adresse 15 wird verwendet, um einen besonderen "Befehlskanal" zu öffnen, über den Zugriff auf den Statusspeicher einer Diskette möglich ist und der zur Ausführung besonderer Operationen dient, die gegen Ende dieses Kapitels unter "Verwaltung von Disketten-Dateien" beschrieben sind.

DER BEFEHLSKANAL IN BASIC<3.0

Wegen seiner Bedeutung verdient der Befehlskanal besondere Erwähnung. In BASIC 4.0 wird ein Befehlskanal automatisch geöffnet, sobald eine Disketten-Datei geöffnet wird, so daß keine besondere Anweisung zum Öffnen eines Befehlskanals erforderlich ist.

In BASIC<3.0 sollten Sie immer vor der Ausführung irgendwelcher Arbeiten mit der Diskette den Befehlskanal öffnen; der Befehlskanal sollte geöffnet bleiben,

bis alle Arbeiten mit der Diskette abgeschlossen sind. Benutzen Sie den Befehlskanal dazu, den Statusspeicher der Diskette abzufragen und besondere Operationen mit der Diskette auszuführen.

ÖFFNEN EINER DISKETTEN-DATEI IN BASIC 4.0

In BASIC 4.0 wird eine Disketten-Datei mit der Anweisung DOPEN# geöffnet. Sie können aber auch die Anweisung OPEN verwenden, da alle Anweisungen aus BASIC 3.0 auch in BASIC 4.0 verwendbar bleiben.

Die Anweisung DOPEN# muß eine logische Dateinummer und einen Dateinamen enthalten. Bei fehlender Angabe wird das Laufwerk 0 des Floppy Disk-Geräts verwendet, es sei denn, Sie geben den Parameter D1 für das Laufwerk 1 an.

Wenn Sie mit dem Parameter LX eine Aufzeichnungslänge angeben, dann wird eine Relative Datei eingerichtet. Sie können von einer Relativen Datei lesen oder sie beschreiben; für die Lese-/Schreiboperationen ist kein besonderer Parameter erforderlich.

Fehlt in der Parameterliste der Anweisung DOPEN# die Angabe der Aufzeichnungslänge LX, dann wird eine Sequentielle Datei eingerichtet. Soll in die Sequentielle Datei geschrieben werden, dann muß ausdrücklich W (für WRITE) in die Parameter-Liste der Anweisung DOPEN# aufgenommen werden; keine Angabe ist erforderlich zum Lesen der Datei.

Für den Floppy Disk-Speicher wird automatisch die Geräteadresse 8 angenommen, es sei denn, Sie geben mit dem Parameter ON UZ eine andere Adresse an. Hier einige Beispiele für die Form von DOPEN#-Anweisungen in BASIC 4.0:

`10 DOPEN#1,"MAIL"`	Öffnen der logischen Datei 1 zum Lesen der Sequentiellen Datei "MAIL" auf Diskette 0.
`50 DOPEN#1,"MAIL",D1,W`	Öffnen der logischen Datei 1 zum Schreiben auf die Sequentielle Datei "MAIL" auf Diskette 1.
`230 DOPEN#5,"DATALIST",D0 ON U5`	Öffnen der logischen Datei 5 zum Lesen der Sequentiellen Datei "DATALIST" auf Diskette 0 des Floppy Disks mit Gerätenummer 5.
`100 DOPEN#2,"MAIL",L100`	Öffnen der logischen Datei 2 zum Zugriff auf die Relative Datei "MAIL" auf Diskette 0. Bei neuer Datei hat jede der Aufzeichnungen eine Länge von 100 Zeichen (Bytes). Bei alter Datei muß die Länge jeder Aufzeichnung bereits beim erstmaligen Öffnen auf 100 Zeichen (Bytes) festgelegt worden sein. Zugriff ist möglich zum Lesen und Schreiben.
`25 DOPEN#3,"SAMPLE",L20,D1`	Öffnen der logischen Datei 3 zum Zugriff auf die Relative Datei "SAMPLE" (Lese- oder Schreiboperation) auf Diskette 1. Bei erstmaligem Öffnen der Datei ist die Länge jeder Aufzeichnung auf 20 Zeichen (Bytes) festgelegt. Bei bestehender Datei muß die Aufzeichnungslänge von 20 Zeichen (Bytes) bereits beim erstmaligen Öffnen definiert worden sein.

Dateinamen können auch durch eine Zeichenketten-Variable statt einer Zeichenkette angegeben werden. Im letzten Beispiel könnte daher auch folgendes geschrieben werden:

```
20 S$="SAMPLE"
25 DOPEN#3,S$,L20,D1
```

ÖFFNEN SEQUENTIELLER DISKETTEN-DATEIEN IN BASIC<3.0

In BASIC<3.0 werden Disketten-Dateien mit der Anweisung OPEN geöffnet. Die untenstehenden OPEN-Anweisungen entsprechen den DOPEN#-Anweisungen zum Öffnen Sequentieller Dateien im letzten Abschnitt. Zur Erinnerung: Relative Dateien können in BASIC<3.0 nicht geöffnet oder bearbeitet werden. Die Sekundären Adressen in folgenden OPEN-Anweisungen wurden willkürlich gewählt:

```
10 OPEN 1,8,2 "MAIL,SEQ"
50 OPEN 1,8,7 "1:MAIL,SEQ,WRITE"
230 OPEN 5,5,3 "0:DATALIST,SEQ"
```

Zeichenketten für den Dateinamen in OPEN-Anweisung können auch durch eine Zeichenketten-Variable ersetzt werden. Die OPEN-Anweisung in Zeile 10 könnte daher auch durch folgende zwei Anweisungen ersetzt werden:

```
5 M$="MAIL,SEQ"
10 OPEN 1,8,2,M$
```

Etwas komplizierter sieht diese Ersetzung für die OPEN-Anweisung in Zeile 50 aus:

```
45 M$="MAIL,SEQ,"
50 OPEN 1,8,7, "1:M$" + "WRITE"
```

FEHLERMELDUNGEN BEIM ÖFFNEN EINER DATEI

Aus folgenden Ursachen können beim Öffnen einer Datei Fehlermeldungen entstehen:

1. Die Fehlermeldung FILE NOT FOUND erscheint, wenn Sie beim erstmaligen Einrichten einer Sequentiellen Datei in der OPEN-Anweisung ein Lesen der Datei vorschreiben. Da eine zu öffnende Datei zunächst leer ist, können aus ihr keine Daten gelesen werden.
2. Die Fehlermeldung FILE TYPE MISMATCH erscheint, wenn Sie eine bestehende Datei öffnen, aber den falschen Dateityp angegeben haben. So etwas geschieht, wenn Sie eine Relative Datei als Sequentielle Datei öffnen, oder wenn Sie eine bestehende Sequentielle Datei als Relative Datei öffnen, oder wenn Sie eine Programm-Datei als Datei für Daten öffnen.
3. Die Fehlermeldung FILE EXISTS erscheint, wenn Sie beim Öffnen einer bestehenden Sequentiellen Datei angeben, daß Sie in diese Datei schreiben wollen. Sie können lediglich in eine neue Sequentielle Datei schreiben.

Wenn Sie in einer OPEN-Anweisung den Namen einer Datei falsch schreiben, dann kann dieser Fehler viel Ärger verursachen, ohne daß es zu einer Warnung in Form einer Fehlermeldung kommt. Das Floppy Disk-Betriebssystem nimmt lediglich an, daß der falsch geschriebene Dateiname eine neue Datei darstellt. Wenn im übrigen das Öffnen einer neuen Datei zulässig war, wird keine Fehlermeldung erzeugt.

SCHLIESSEN EINER DISKETTEN-DATEI

In BASIC 4.0 verwenden Sie zum Schließen einer Datei auf Disketten die Anweisung:

```
DCLOSE#N
```

und in BASIC<3.0 die Anweisung:

```
CLOSE N
```

wobei die logische Dateinummer N als erster Parameter in den Anweisungen OPEN oder DOPEN# verwendet wurde.

Das Schreiben in eine Disketten-Datei muß mit einer DCLOSE#- oder CLOSE-Anweisung abgeschlossen werden, da sonst einige der letzten Daten nicht die Diskette erreichen könnten (sondern im Pufferspeicher hängen bleiben).

Das Schließen der Datei nach einem Schreibvorgang ist zwar nicht erforderlich, wird aber aus Gründen einer guten Programmierpraxis getan.

Der Computer schließt automatisch alle offenen Dateien, sobald die END-Anweisung des Programms ausgeführt wird. Hierbei wird angenommen, daß die Floppy Disk-Speicher noch eingeschaltet sind. Trotz allem ist es gute Programmierpraxis, die Dateien individuell mit CLOSE-Anweisungen zu schließen, statt hierfür die am Programmende stehenden END-Anweisungen zu benutzen. Dieses Thema wurde ausführlich für Dateien auf Bandkassetten am Kapitelanfang diskutiert. Das Schließen von Dateien auf Bandkassette kann auch auf Dateien auf Disketten übertragen werden.

FEHLERMELDUNG UND GERÄTESTATUS BEI FLOPPY DISK-SPEICHERN

Alle CBM Floppy Disk-Speicher verfügen über ein rotes Warnlicht zur Fehleranzeige. Das Warnlicht leuchtet auf, wenn eine Benutzung der Diskette nicht erfolgreich möglich war. Eine weitere Benutzung der Diskette ist nicht möglich, solange das Warnlicht brennt. Zum Löschen des Warnlichts muß der Programmlauf durch Drücken der Taste STOP unterbrochen und das Statusregister des Floppy Disk-Speichers zur Fehlerinterpretation gelesen werden.

Ein guter Vorschlag ist, nach jeder Operation mit der Diskette das Statusregister des Floppy Disk-Speichers zu lesen und in Programmen für den Datenaustausch mit Disketten Prüfroutinen für das Statusregister vorzusehen.

Erinnern Sie sich aus Kapitel 1, daß Disketten vor ungewolltem Überschreiben Ihres Inhalts durch Überkleben des seitlichen Schlitzes in der Schutzhülle geschützt werden. Eine solche Diskette nennt man schreib-geschützt. Wenn Sie die Übertragung einer Datei auf eine schreib-geschützte Diskette versuchen, dann bleibt das Programm im CBM-Computer hängen. Der Computer versucht unentwegt, die Diskette zu beschreiben, erhält aber von der Diskette keine Rückmeldung in Form eines Fehlersignals. Diese Situation zeigt sich darin, daß der Computer nicht zu arbeiten scheint und der Programmablauf nicht durch Drücken der Taste STOP unterbrochen werden kann. Wenn dieser Fall eintritt, müssen Sie die Diskette aus dem Floppy Disk-Speicher herausnehmen, den CBM-Computer ausschalten und den Betrieb erneut beginnen.

BEHANDLUNG VON FLOPPY DISK-FEHLERMELDUNGEN IN BASIC 4.0

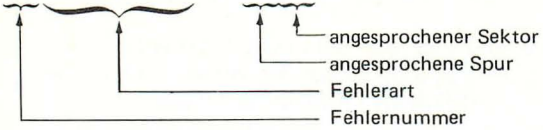
In BASIC 4.0 können Sie die Fehleranzeige eines Floppy Disk-Speichers im Statusregister DS in direktem Dialog mit dem Computer oder im Rahmen eines Programms löschen.

Im direkten Dialog geben Sie einen PRINT-Befehl zur Darstellung der Variablen DS oder der Zeichenketten-Variablen DS$ auf dem Bildschirm ein.

Die numerische Variable DS stellt die Fehlermeldung im Statusregister als dezimalverschlüsselte Zahl dar, deren Interpretation mit Tabelle 6-2 erfolgt.

Die Zeichenketten-Variable DS$ stellt folgende vier Parameter auf dem Bildschirm dar:

```
?DS$
NN FEHLERMELDUNG TT SS
```



Anhang B enthält eine Übersicht der Fehlermeldungen eines Floppy Disk-Speichers.

Programme in BASIC 4.0 sollten zur Fehlererkennung das Status-Register DS wie folgt abfragen:

```
20 IF DS <> 0 THEN PRINT "FEHLER"
```

Bei Vorliegen einer Fehlermeldung löscht diese Anweisung das Warnlicht und das Statusregister, unterbricht den Programmlauf und gibt folgende Nachricht auf dem Bildschirm aus:

```
ERROR
BREAK IN XXXX
```

Mit XXXX wird die Zeilennummer angegeben, in der die Programmunterbrechung erfolgte. Mehr Informationen über den Fehler erhält der Benutzer durch Darstellung der Variablen DS$. Hierfür lautet die Anweisung wie folgt:

```
20 IF DS <> 0 THEN PRINT DS$:STOP
```

Bei Vorliegen einer Fehlermeldung löscht diese Anweisung das Warnlicht und das Statusregister, unterbricht den Programmlauf und stellt folgende Nachricht auf dem Bildschirm dar:

```
NN FEHLERMELDUNG TT SS
BREAK IN XXXX
READY
```

In Anhang B finden Sie eine Erklärung der Fehlernummern NN. Mit TT wird die Spur und mit SS der Sektor angegeben, auf die beim Eintreten des Fehlers zugegriffen worden war.

BEHANDLUNG VON FLOPPY DISK-FEHLERMELDUNGEN IN BASIC<3.0

In BASIC<3.0 haben Sie keinen Zugriff auf die Variablen DS oder DS$. Zur Fehlerinterpretation müssen Sie mit einer OPEN-Anweisung eine logische Datei öffnen, in der die Gerätenummer 8 mit der Sekundären Adresse 15 angegeben ist. Sie müssen dann vier Zeichenketten-Variablen eingeben und mit einer PRINT-Anweisung deren Inhalt zur Fehlerinterpretation auf dem Bildschirm darstellen. Als Teil eines Programms sieht das wie folgt aus:

```
10 OPEN 1,8,15
20 INPUT#1,A$,B$,C$,D$
30 PRINT A$,B$,C$,D$
40 CLOSE 1
```

Die Anweisung INPUT#1 in Zeile 20 kann nicht in direktem Dialog mit dem Computer verwendet werden. Die Fehlerinterpretation ist in A$, B$, C$ und D$

wie folgt enthalten: Fehlernummer A$, Fehlermeldung B$, Spurnummer C$, Sektornummer D$. Diese Melsungen entsprechen denjenigen, die im vorigen Kapitel genannt wurden. Den Zeichenketten-Variablen A$, B$, C$ und D$ in Zeilen 20 und 30 kann auch jeder andere Name gegeben werden.

Programme in BASIC<3.0 sollten jeden Zugriff auf eine Diskette mit einer Abfrage des Floppy Disk-Speichers nach Fehlermeldungen verbinden. Hierzu wird vor Zugriff auf die Diskette eine logische Datei geöffnet,deren OPEN-Anweisung die Geräteadresse und die Sekundäre Adresse 15 enthält. Nach dem Zugriff auf die Diskette wird die Fehlernummer A$ geprüft: ist diese Nummer 0, dann war der Zugriff auf die Diskette fehlerfrei. Die Programmlogik hierfür sieht wie folgt aus:

```
10 OPEN 15,8,15
.
150  (BEFEHLE FUER DEN ZUGRIFF AUF DISKETTTEN)
.
160 REM PRUEFUNG DES BETRIEBSZUSTANDS DER DISKETTE
170 INPUT#15,A$,B$,C$,D$
180 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$:STOP
.
.
```

Enthält ein Programm zahlreiche Zugriffe auf eine Diskette, dann wiederholen sich die Anweisungen in Zeilen 170 und 180 in der Befehlsfolge des Programms. Sie könnten versucht sein, aus diesen Anweisungen ein Unterprogramm zu machen. Das ist zwar möglich, Sie verlieren aber den Hinweis, an welcher Stelle im Programm die Fehlermeldung erfolgte, da die STOP-Anweisung immer die gleiche Zeilennummer im Unterprogramm als Ort der Programmunterbrechung meldet. Werden im Gegensatz dazu die Anweisungen aus Zeilen 170 und 180 im Programmm dort wiederholt, wo Zugriffe auf die Diskette erfolgen, dann können Sie durch Blick auf die Zeilennummer bei Programmunterbrechung durch die STOP-Anweisung erkennen, wo im Programm sich ein fehlerhafter Zugriff auf die Diskette ereignete.

VERWALTUNG VON DISKETTEN-DATEIEN

Außer dem Lesen und Schreiben von Dateien sind in BASIC folgende Operationen möglich:

1. Anlegen einer neuen Diskette.
2. Löschen einer alten Diskette und Vorbereiten für die Wiederverwendung.
3. Darstellen des Inhaltsverzeichnisses einer Diskette.
4. Überprüfen der Diskette auf unbenutzten Speicherplatz (nur in BASIC 4.0).
5. Kopieren einer Datei.
6. Kopieren einer Diskette.
7. Neubenennung einer Datei (nur in BASIC 4.0).
8. Löschen einer Disketten-Datei oder Ersetzen ihres Inhalts.

In BASIC<3.0 beginnt jede Operation mit einer Diskette mit der Anweisung OPEN. Sie können dann lesen, schreiben oder eine der verwaltungstechnischen Opera-

tionen ausführen, die oben beschrieben wurden. Die Operation endet mit einer CLOSE-Anweisung. In BASIC 4.0 wird eine Datei vor dem Lesen oder Schreiben mit OPEN geöffnet und danach mit CLOSE geschlossen. Die oben beschriebenen verwaltungstechnischen Operationen jedoch werden über besondere Anweisungen ausgeführt, die nicht zwischen OPEN- und CLOSE-Anweisungen stehen müssen.

Wir werden alle diese verwaltungstechnischen Operationen beschreiben, ehe wir uns die Programmlogik zum Arbeiten mit Dateien ansehen.

Obwohl verwaltungstechnische Operationen häufig in direktem Dialog mit dem Computer ausgeführt werden, können sie auch im Rahmen eines Programms erfolgen. Kapitel 8 enthält eine vollständige Beschreibung der BASIC-Anweisungen für die Verwaltung von Disketten und Dateien. Wenn Sie mit dem nachfolgenden Text Schwierigkeiten haben, weil Sie eine BASIC-Anweisung nicht verstehen, dann lesen Sie die Beschreibung zu dieser BASIC-Anweisung in Kapitel 8 und setzen dann die Lektüre fort.

ANLEGEN UND INITIIEREN EINER DISKETTE

Eine unbenutzte Diskette kann nicht einfach in einen Floppy Disk-Speicher eingelegt und beschrieben werden. Die Oberfläche der Diskette muß zunächst vorbereitet werden. Auf den Spuren der Diskette müssen die Sektoren markiert, das Inhaltsverzeichnis geschrieben und die Block-Belegungsliste abgelegt werden. Überdies muß die Diskette einen Namen erhalten. Sie können auch eine benutzte Diskette neu anlegen; hierdurch werden alle früheren Daten gelöscht und die Diskette zur Wiederverwendung vorbereitet.

Im allgemeinen erfolgt das Anlegen einer Diskette in direktem Dialog mit dem Computer.

ANLEGEN EINER DISKETTE IN BASIC 4.0

In BASIC 4.0 legen Sie eine neue Diskette mit der Anweisung HEADER an:

HEADER "DISKETTEN-NAME", DX, IYY

Der "DISKETTEN-NAME" kann aus bis zu 16 Symbolen bestehen. X gibt die Nummer des Laufwerks an, in dem sich die Diskette befindet und muß den Wert 0 oder 1 haben. YY ist eine Zahl, die Sie der Diskette zur Identifizierung geben müssen.

Das Anlegen einer Diskette dauert ungefähr 2 Minuten. Wenn das Anlegen der Diskette aus irgendeinem Grund nicht möglich ist, erscheint folgende Nachricht auf dem Bildschirm:

$BAD DISK

Diese Nachricht kann einen der folgenden Gründe haben:

1. Sie haben vergessen, die Diskette in das angegebene Laufwerk zu laden.
2. Sie haben in der HEADER-Anweisung die falsche Laufwerknummer angegeben.

3. Sie haben vergessen, in der Parameter-Liste der HEADER-Anweisung eine Diskettennummer anzugeben.
4. Die Diskette ist schreib-geschützt (der seitliche Schlitz in der Schutzhülle ist geschlossen)
5. Die Diskette hat eine fehlerhafte magnetische Oberfläche.

Beim Anlegen einer benutzten Diskette brauchen Sie nur noch die Laufwerk-Nummer in der Parameter-Liste der HEADER-Anweisung anzugeben. Der alte Name der Diskette wird beibehalten, wenn Sie keinen neuen Namen eingeben. Eine alte Disketten-Nummer wird durch eine neue Angabe ersetzt; erfolgt keine neue Angabe, dann wird die alte Nummer behalten. Beachten Sie aber: Sie können keine neue Disketten-Nummer angeben, ohne der Diskette auch einen neuen Namen zu geben. Wenn Sie es dennoch versuchen, erhalten Sie eine Fehlermeldung (SYNTAX-Fehler).

In BASIC 4.0 wird angenommen, daß der Floppy Disk-Speicher die Gerätenummer 8 trägt. Wenn Sie aus irgendeinem Grunde für den Floppy Disk-Speicher eine andere Gerätenummer verwenden, dann müssen Sie beim Anlegen der Diskette diese Information in der Parameter-Liste der HEADER-Anweisung mit ON UZ oder UZ angeben, worin Z die gewünschte Gerätenummer ist.

Das Anlegen einer benutzen Diskette dauert nur einige Sekunden.

Die HEADER-Anweisung kann in folgenden Formen erfolgen. Rückmeldungen auf dem Bildschirm wurden nicht angegeben:

```
HEADER "MUSTER",D0,I01
```
Vorbereitung der Diskette in Laufwerk 0 unter dem Namen "MUSTER" und der Nummer 01.

```
HEADER D0
```
Vorbereitung einer alten Diskette in Laufwerk 0 unter gleichem Namen/Nummer.

```
HEADER "NEU",D1
```
Vorbereitung einer Diskette in Laufwerk 1 unter neuem Namen "NEU" und gleicher Nummer.

```
HEADER "MUSTER",D0,I05,ON U7
```
Vorbereitung einer Diskette in Laufwerk 0 eines Floppy Disks mit Gerätenummer 7. Der Name ist "MUSTER" und die Nummer 05.

```
HEADER D1,I01
?SYNTAX ERROR
```
Die Anweisung HEADER wird nicht ausgeführt, da eine neue Disketten-Nummer ohne neuen Disketten-Namen definiert wird.

ANLEGEN EINER DISKETTE IN BASIC<3.0

In BASIC<3.0 müssen Sie zum Anlegen einer Diskette den Befehlskanal mit OPEN öffnen und dann eine PRINT#-Anweisung ausführen, die die gleiche logische Dateinummer wie in der Parameterliste der OPEN-Anweisung verwendet. Die PRINT#-Anweisung muß folgende Zeichenkette, eingeschlossen in Anführungszeichen, enthalten:

"NEWX: DISKETTEN-NAME, YY"

NEW kann durch N ersetzt werden. X ist die Laufwerknummer; sie muß den Wert 0 oder 1 haben. Mit DISKETTEN-NAME erhält die Diskette einen Namen, der aus bis zu 16 Symbolen bestehen darf. YY ist die Nummer der Diskette. Die OPEN-Anweisung zum Öffnen des Befehlskanals kann jede logische Dateinummer enthalten, muß aber als Gerätenummer den Wert 8 und als sekundäre Adresse den Wert 15 angeben.

Hier einige Beispiele der genannten OPEN- und PRINT#-Anweisungen:

```
OPEN 1,8,15
PRINT#1, "N0:MUSTER,01"
```

Eine Diskette in Laufwerk 0 wird unter dem Namen "MUSTER" mit der Nummer 01 initiiert.

```
OPEN 3,8,15
PRINT#3, "NEW1:NEU,01"
```

Eine Diskette in Laufwerk 1 wird unter dem Namen "NEU" mit der Nummer 01 initiiert.

Das Anlegen von Disketten mit Anweisungen aus BASIC<3.0 ist nicht immer auf CBM-Computern möglich, die mit BASIC 4.0 arbeiten. Manchmal setzt das Laufwerk die Umdrehung der Diskette auch nach Abschluß der Disketten-Initiierung fort.

In BASIC<3.0 ist das Anlegen einer alten Diskette möglich, wodurch alle früheren Speicherungen auf der Diskette gelöscht werden und die Oberfläche für die Wiederverwendung vorbereitet wird. Beim Anlegen einer alten Diskette müssen Sie keine neue Disketten-Nummer in der Parameter-Liste PRINT#-Anweisung angeben; bei fehlender neuer Nummer wird die alte Disketten-Nummer verwendet.

INITIIEREN EINER DISKETTE IN BASIC<3.0

In BASIC<3.0 muß eine beschriebene Diskette vor dem Öffnen einer Datei initiiert werden. Zum Initiieren der Diskette öffnen Sie den Befehlskanal, gefolgt von einer PRINT#-Anweisung, in deren Parameter-Liste als Zeichenketten-Variable der Buchstabe I oder das Wort INITIALIZE sowie die Nummer des Laufwerks im Floppy Disk-Speicher erscheint. Die Laufwerk-Nummer kann ausgelassen werden, wodurch die Disketten in beiden Laufwerken initiiert werden.

Disketten werden für gewöhnlich im Rahmen eines Programms initiiert. Beim Initiieren einer Diskette werden Daten auf der Oberfläche der Diskette nicht geändert.

Hier einige Beispiele für Initiierungs-Anweisungen in BASIC<3.0:

```
10 OPEN 1,8,15
20 PRINT#1,"I0"
```

Initieren der Diskette 0

```
5 OPEN 3,8,15
10 PRINT#1, "INITIALIZE"
```

Initiieren der Disketten 0 und 1

Eine gerade angelegte Diskette muß nicht initiiert werden. Das Anlegen einer Diskette schließt ihre Initiierung ein.

AUSGABE DES DATEIEN-VERZEICHNISSES

AUSGABE DES DATEIEN-VERZEICHNISSES IN BASIC 4.0

Vor dem Zugriff auf eine Datei ist es ratsam, das Inhaltsverzeichnis der Diskette auf dem Bildschirm darzustellen. In BASIC 4.0 wird dies mit der Anweisung DIRECTORY getan. Die DIRECTORY-Anweisung wird gewöhnlich in direktem Dialog mit dem Computer ausgeführt. Hier einige Beispiele für die Form dieser Anweisung:

```
DIRECTORY                    Darstellen des Dateiverzeichnisses beider Disketten
DIRECTORY D1                 Darstellen des Dateiverzeichnisses von Diskette 1
DIRECTORY D1 ON U8

DIRECTORY D0                 Darstellen des Dateiverzeichnisses von Diskette 0
```

Statt DIRECTORY kann das Wort CATALOG benutzt werden. Das Dateien-Verzeichnis wird wie folgt im Bildschirm dargestellt:

```
0 "DISKETTENNAME NN XX"
Y     "DATEINAME"          TYP
Y     "DATEINAME"          TYP

ZZZZ BLOCKS FREE           Y
```

Name und Nummer der Diskette erscheinen in Negativschrift am Bildschirmanfang. NN ist die Nummer der Diskette, XX die Version des Disketten-Betriebssystems. Danach folgt eine Auflistung der Namen von Dateien, die auf der Diskette gespeichert sind. Links vom Dateinamen steht die Anzahl der Sektoren, die der Datei zugeordnet wurden. Rechts vom Dateinamen steht der Dateityp: REL für eine Relative Datei, SEQ für eine Sequentielle Datei oder PRG für eine Programmm-Datei. Schließlich wird die Zahl der unbenutzten Sektoren (Blöcke) dargestellt. Über die ebenfalls dargestellten Benutzer-Dateien siehe Kapitel 7.

Beim Arbeiten mit der Anweisung DIRECTORY müssen beide Laufwerke mit einer Diskette geladen sein. Ein häufiger Fehler entsteht beim Eingeben der Anweisung DIRECTORY, wenn das Dateien-Verzeichnis der Diskette in Laufwerk 0 gewünscht, aber in Laufwerk 1 keine Diskette geladen ist. Es erfolgt eine Fehlermeldung, und die Darstellung der Inhaltsverzeichnisse unterbleibt. Erinnern Sie sich an das Löschen der Warnlampe nach erfolgter Fehlermeldung durch Lesen des Status-Registers mit der Anweisung ? DS$ RETURN. Sie können den Floppy Disk-Speicher erst nach Löschen der Warnlampe wieder benutzen. Eine Fehleranzeige erfolgt ebenfalls, wenn Sie in der DIRECTORY-Anweisung das falsche Laufwerk angeben. Ein Beispiel: Wenn Laufwerk 1 mit einer Diskette geladen ist, Sie aber in direktem Dialog folgende Anweisung eingeben:

```
DIRECTORY D0
```

dann erhalten Sie eine Fehleranzeige, aber keine Darstellung des Dateien-Verzeichnisses.

DARSTELLUNG DES DISKETTEN-VERZEICHNISSES BASIC<3.0

In BASIC<3.0 bringen Sie das Inhaltsverzeichnis mit Hilfe einer LOAD-Anweisung zur Anzeige:

```
LOAD "$X",Y
```

X ist die Nummer des Laufwerks (0 oder 1) und Y die Gerätenummer (gewöhnlich 8). Danach folgt der Standard-Dialog für das Laden von Programmen. Nach Laden des Programmes listen Sie es auf, um das Inhaltsverzeichnis auf dem Bildschirm darzustellen. Im folgenden Beispiel wird das Inhaltsverzeichnis für eine Diskette im Laufwerk 0 dargestellt:

```
LOAD "$0", 8
SEARCHING FOR $0
LOADING
READY
LIST
```

INVENTUR EINES DATEIEN-VERZEICHNISSES

In BASIC 4.0 gibt es die Anweisung COLLECT, mit der Sie auf einer Diskette "Hausputz" machen können.

Die COLLECT-Anweisung ermittelt Sektoren, die von Dateien belegt, aber nicht benutzt wurden. Diese Sektoren werden wieder zugänglich gemacht, und das Inhaltsverzeichnis der Diskette wird entsprechend geändert.

Die COLLECT-Anweisung wird gewöhnlich in direktem Dialog mit dem Computer ausgeführt.

`COLLECT` *Inventarisierung beider Disketten*

`COLLECT D0` *Inventarisierung der Diskette O*

Einige Versionen von BASIC 4.0 haben Probleme mit der Anweisung SCRATCH, die die Löschung von unvorschriftsmäßig geschlossenen Dateien verweigert. Wenn Sie vor der SCRATCH-Anweisung eine COLLECT-Anweisung ausführen, können Sie Probleme mit SCRATCH auf Ihrem CBM-Computer vermeiden. Die unvorschriftsmäßig geschlossene Datei kann dann ohne Probleme durch die SCRATCH-Anweisung gelöscht werden.

ÜBERSPIELEN VON DATEIEN UND GANZEN DISKETTEN

Sie sollten Ersatzkopien von Dateien anfertigen, die Sie ständig bewahren wollen. Wenigstens eine Kopie der Datei sollte sich auf einer anderen Diskette befinden. Eine Kopie der Datei auf der gleichen Diskette ist keine Abhilfe gegen eine zufällige Löschung der gesamten Diskette.

CBM-BASIC-Anweisungen ermöglichen Ihnen, einzelne Dateien oder den Inhalt einer ganzen Diskette auf eine andere Diskette zu überspielen.

ÜBERSPIELEN EINER DATEI IN BASIC 4.0

In BASIC 4.0 ermöglicht die Anweisung COPY, eine einzelne Datei oder eine ganze Diskette zu überspielen. Jedoch läßt die COPY-Anweisung nur eine Gerätenummer zu, so daß Überspielungen zwischen Disketten in den beiden Laufwerken eines Floppy Disk-Speichers erfolgen müssen.

Wird in der COPY-Anweisung ein Dateiname angegeben, dann wird eine einzelne Datei überspielt. Fehlt die Angabe eines Dateinamens, dann werden alle Dateien

auf der Diskette überspielt. Hier einige Formen der COPY-Anweisung für den direkten Dialog mit dem Computer:

```
COPY D0 TO D1
```
Kopieren aller Dateien von Diskette 0 auf Diskette 1

```
COPY D0,"TESTDATEN" TO D1,"TESTDATEN"
```
Kopieren der Datei TESTDATEN von Diskette 0 auf Diskette 1 unter gleichem Dateinamen

```
COPY D1,"TESTDATEN" TO D0,"NEUDATEN"
```
Kopieren der Datei TESTDATEN von Diskette 0 auf Diskette 1 unter neuem Dateinamen NEUDATEN

ÜBERSPIELEN VON DATEIEN IN BASIC < 3.0

In BASIC < 3.0 benutzen Sie die Anweisung PRINT# mit folgender Zeichenkette als Parameter:

"COPYM: NEUERNAME = N: ALTERNAME"

Statt COPY können Sie auch den Buchstaben C schreiben. N ist die Laufwerknummer mit der Diskette, auf der sich die Original-Datei befindet. ALTER NAME stellt den Namen der Original-Datei dar. M ist die Nummer des Laufwerks, das mit der Diskette für die überspielte Datei geladen ist; NEUER NAME ist der Name der überspielten Datei.

Hier einige Beispiele:

```
OPEN 15,8,15
PRINT#15,"COPY1:ADRESSEN=0:ADRESSEN"
CLOSE 15
```
Kopieren der Datei ADRESSEN von Diskette 0 auf Diskette 1 unter gleichem Dateinamen

```
OPEN 15,8,15
PRINT#15,"C0:TESTDATEN=1:NEUDATEN"
CLOSE 1
```
Kopieren der Datei TESTDATEN von Diskette 0 auf Diskette 1 unter neuem Dateinamen NEUDATEN

VERKETTUNG VON DATEIEN IN BASIC < 3.0

Anläßlich der Überspielung von Dateien ermöglicht die PRINT#-Anweisung in BASIC < 3.0, zwei, drei oder vier Original-Dateien zu einer einzigen Datei zu verketten. Im folgenden Beispiel werden in direktem Dialog mit dem Computer die Dateien DATA 1 und DATA 2 von der Diskette im Laufwerk 0 auf die Diskette im Laufwerk 1 überspielt und dabei verkettet, wobei der verketteten Datei der Name DATAX gegeben wird:

```
OPEN 15,8,15
PRINT#15,"C1:DATAX=1:DATA1,1:DATA2"
CLOSE 15
```

Verkettete Original-Dateien müssen nicht auf der gleichen Diskette stehen. In unserem Beispiel könnte die verkettete Datei DATAX von Dateien stammen, die auf der gleichen oder zwei verschiedenen Disketten stehen.

FEHLERMELDUNGEN BEIM ÜBERSPIELEN VON DATEIEN

Die überspielte Datei darf nicht den Namen einer bereits existierenden Datei erhalten. Ist dies der Fall, dann wird die COPY-Anweisung nicht ausgeführt. Das

Warnlicht am Floppy Disk-Speicher leuchtet auf; wenn Sie die Fehlermeldung im Statusregister lesen, werden Sie die Meldung FILE EXIST vorfinden.

Beim Überspielen einer Datei von einer Diskette auf eine andere in BASIC 4.0 mit der Anweisung COPY wird die Ausführung der Anweisung unterbrochen, wenn der Name der Original-Datei auf derjenigen Diskette vorgefunden wird, auf die die Überspielung erfolgt. Das Warnlicht am Floppy Disk-Speicher leuchtet auf.

ÜBERSPIELEN EINER DISKETTE

Sie können Dateien einer Diskette auf eine andere überspielen; Sie können auch eine Ersatz-Diskette durch Duplizieren schaffen. Das sind zwei verschiedene Vorgänge. Die Ersatz-Diskette stellt ein genaues Duplikat der Original-Diskette dar, mit dem gleichen Disketten-Namen, der gleichen Disketten-Nummer und den gleichen Datei-Namen. Wenn Sie dagegen alle Dateien einer Original-Diskette auf eine neue Diskette überspielen, dann ändert sich weder der Name der neuen Diskette, noch die Namen derjenigen Dateien, die sich vor der Überspielung bereits auf der neuen Diskette befanden. Die neue Diskette hat also einen anderen Namen als die Original-Diskette, und obwohl sie alle Dateien der Original-Diskette enthält, kann sie auch aus Dateien bestehen, die auf der Original-Diskette nicht enthalten waren.

DUPLIZIEREN EINER DISKETTE IN BASIC 4.0

In BASIC 4.0 benutzen Sie die Anweisung BACK UP, um eine Diskette zu duplizieren. Die Überspielung erfolgt zwischen Disketten in den Laufwerken 0 und 1 eines Floppy Disk-Speichers mit beliebiger zulässiger Gerätenummer. Hier einige Beispiele der BACK UP-Anweisung für den direkten Dialog mit dem Computer:

```
BACKUP D0 TO D1
```
Diskette in Laufwerk 0 kopieren auf Diskette in Laufwerk 1

```
BACKUP D1 TO D0 ON U5
```
Diskette in Laufwerk 1 kopieren auf Diskette in Laufwerk 0
Das Floppy Disk trägt die Gerätenummer 5

Die BACK UP-Anweisung ermöglicht die Überspielung auf eine Diskette, die nicht initiiert oder angelegt wurde. Wenn erforderlich, erfolgt vor Überspielung mit BACK UP automatisch ein Anlegen der Diskette.

DUPLIZIEREN EINER DISKETTE IN BASIC < 3.0

In BASIC < 3.0 dient die Anweisung PRINT# zur Duplizierung einer Diskette. Die Duplizierung erfolgt zwischen Disketten in den Laufwerken 0 und 1 eines Floppy Disk-Speichers. Die Duplizierung ist nicht möglich für Disketten in verschiedenen Floppy Disk-Speichern. Die PRINT#-Anweisung muß folgende Zeichenkettenvariable in ihrer Parameterliste enthalten:

```
"DUPLICATEN=M"
```

Statt DUPLICATE können Sie auch den Buchstaben D verwenden. N ist die Laufwerknummer derjenigen Diskette, auf die die Überspielung erfolgt. M ist die Laufwerknummer der Original-Diskette.

Hier das Beispiel einer Duplizierung in direktem Dialog mit dem Computer:

```
OPEN 15,8,15
PRINT#15, "D1=0"
CLOSE 15
```

Kopiere Diskette 0 auf Diskette 1

NEUBENENNUNG EINER DATEI

Jede Datei für Programme oder Daten kann neu benannt werden. Am häufigsten werden Dateien für Programme im Verlauf der Erstellung und Korrektur eines Programmes neu benannt.

NEUBENENNUNG EINER DATEI IN BASIC 4.0

In BASIC 4.0 dient die Anweisung RENAME der Neubenennung einer einzelnen Datei. Hier ein Beispiel für den direkten Dialog mit dem Computer:

```
RENAME D0, "SEQ.NUM.B4" TO "SEQNUM"
```

NEUBENENNUNG EINER DATEI IN BASIC < 3.0

In BASIC < 3.0 erfolgt die Neubenennung einer Datei mit der Anweisung PRINT# und den folgenden Zeichenkettenvariablen in der Parameterliste:

"RENAMEX: NEUERNAME = ALTERNAME"

Statt RENAME genügt auch der Buchstabe R. X ist die Laufwerknummer derjenigen Diskette, auf der die neubenannte Datei gespeichert ist. NEUER NAME ist der neue Dateiname; er ersetzt ALTER NAME, den ursprünglichen Namen der Datei.

Hier ein Beispiel:

```
OPEN 15,8,15
PRINT#15, "R0:SEQNUM=SEQ.NUM.B4"
CLOSE 15
```

Die Datei SEQ. NUM.B4 in Laufwerk 0 wird unbenannt in SEQNUM

LÖSCHEN VON DATEIEN

Jede Datei auf einer Diskette kann gelöscht werden. Wenn Sie eine Datei in direktem Dialog mit dem CBM-Computer löschen wollen, erscheint zuerst die Frage ARE YOU SURE? auf dem Bildschirm. Hierauf müssen Sie durch Eingabe von "YES" und RETURN antworten, ehe die Datei gelöscht wird. Erfolgt die Löschung als Teil eines Programms, dann erscheint keine Frage auf dem Bildschirm.

LÖSCHEN EINER DATEI IN BASIC 4.0

In BASIC 4.0 können Sie eine Disketten-Datei mit der Anweisung SCRATCH löschen. In direktem Dialog mit dem Computer wird diese Anweisung wie folgt eingegeben:

```
SCRATCH D0, "REL.NUM.B4"  Löschen der Datei REL.NUM.B4 in Laufwerk 0
```

Die gleiche Anweisung kann auch innerhalb eines Programms vorkommen:

```
.
.
240 DCLOSE
250 SCRATCH D0, "REL.NUM.B4"
.
.
```

Einige Versionen von BASIC 4.0 haben Probleme mit der SCRATCH-Anweisung, da die Löschung nicht vorschriftsmäßig geschlossener Dateien unterbleibt. Diese Probleme verschwinden, wenn erst eine Inventur der Diskette mit der Anweisung COLLECT erfolgt, und danach die Löschung mit SCRATCH.

LÖSCHEN EINER DATEI IN BASIC < 3.0

In BASIC < 3.0 können eine oder mehrere Dateien mit einer einzigen PRINT#-Anweisung gelöscht werden. In der Parameter-Liste PRINT#-Anweisung muß folgende Zeichenkettenvariable stehen:

"SCRATCHX: DATEINAME"

Statt SCRATCH können Sie auch ein S benutzen. Die Diskette mit der zu löschenden Datei soll sich im Laufwerk X des Floppy Disk-Speichers befinden. DATEINAME ist der Name der zu löschenden Datei. In direktem Dialog mit dem Computer sieht die Löschung einer einzelnen Datei wie folgt aus:

```
OPEN 15,8,15
PRINT#15, "S0:REL.NUM.B4"  Lösche Datei REL.NUM.B4 auf Diskette 0
CLOSE 15
```

Zur Löscung von zwei oder mehr Dateien erweitern Sie einfach die Zeichenkette in der PRINT#-Anweisung um weitere Laufwerk-Nummern und Dateinamen. Ein Beispiel: Sie können oben genannte Anweisungen abwandeln und zwei Dateien wie folgt löschen:

```
OPEN 15,8,15                                  Lösche Datei REL.NUM.B4 auf Diskette 0 und
PRINT#15, "S0:REL.NUM.B4,1:REL.NUM.B<3"       Datei REL.NUM.B<3 auf Diskette 1
CLOSE 15
```

Wenn Sie einen oder mehrere Buchstaben eines Dateinamens mit einem Stern-Symbol abschließen, dann wird jede Datei gelöscht, in deren Name die vor dem Stern-Symbol stehenden Buchstaben vorkommen. Sehen Sie folgendes Beispiel:

```
OPEN 15,8,15
PRINT#15, "S0:NUM*"
CLOSE 15
```

Jede Datei auf der Diskette im Laufwerk 0, deren Name mit den drei Buchstaben NUM beginnt, wird gelöscht.

Wenn Sie im Namen einer Datei eines der Symbole durch ein Fragezeichen ersetzen, dann können zu löschende Dateien an dieser Stelle ein beliebiges Symbol in Ihrem Namen haben. Ein Beispiel: Die folgenden Anweisungen löschen eine Datei, deren Name mit NUM beginnt, mit SEQ endet und dazwischen aus vier Symbolen besteht:

```
OPEN 15,8,15
PRINT#15, "S0:NUM????.SEQ"
CLOSE 15
```

SEQUENTIELLE DATEIEN FÜR DATEN

BASIC 4.0 und BASIC < 3.0 ermöglichen das Einrichten Sequentieller Dateien zum Speichern von Daten.

Eine Sequentielle Daten-Datei wird entweder für einen Schreibvorgang oder für einen Lesevorgang geöffnet, niemals für beides zugleich. Der Vorgang des Öffnens einer Datei dient auch zum Einrichten einer neuen Sequentiellen Datei. Die neue Sequentielle Datei kann nur für einen Schreibvorgang geöffnet werden; sie kann nicht für einen Lesevorgang geöffnet werden. Eine schon bestehende Sequentielle Datei kann dagegen nur für einen Lesevorgang geöffnet werden; sie darf nicht für einen Schreibvorgang geöffnet werden.

TRENNSYMBOLE FÜR FELDER IN SEQUENTIELLEN DATEIEN

Numerische Variable müssen in einer Sequentiellen Datei durch Wagenrücklauf-Symbole abgeschlossen werden. Zeichenketten-Variable können durch Komma-Symbole oder durch Wagenrücklauf-Symbole abgeschlossen werden.

Wir empfehlen bei der Speicherung von Daten die Verwendung des Wagenrücklauf-Symbols zur Trennung aller Felder in einer Sequentiellen Datei. Die Verwendung des Komma-Symbols zur Trennung von Zeichenketten-Variablen bietet keinen erkennbaren Vorteil und kann nur unnötige Probleme bei der Programmierung verursachen.

Wenn alle Felder mit einem Wagenrücklauf-Symbol abgeschlossen sind, dann vereinfachen sich die Regeln für das Schreiben in eine Sequentielle Daten-Datei: benutzen Sie die PRINT#-Anweisung mit einer Parameterliste, wie man sie in einer PRINT-Anweisung zur Bildschirmdarstellung von Variablen in einer vertikalen Spalte verwenden würde. Das Lesen der Daten von der Datei erfolgt mit den Anweisungen INPUT# oder GET#. In BASIC 4.0 und mit dem Disketten-Betriebssystem DOS 2.0 werden Eingaben mit PRINT#-Anweisungen automatisch mit einem Wagenrücklauf-Symbol abgeschlossen, sofern als logische Dateinummer der Wert 128 oder höher eingesetzt wird. Der Abschluß einer Variablen durch ein Wagenrücklauf-Symbol unterbleibt, wenn als logische Dateinummer der Wert 127 oder geringer eingesetzt wird.

SCHREIBEN NUMERISCHER DATEN IN EINE SEQUENTIELLE DATEI

Zu Anfang ein sehr einfaches Programm-Beispiel, das eine Sequentielle Datei öffnet und in der Datei zehn Aufzeichnungen speichert, von denen jede aus 10 Zah-

len besteht, wie folgendes Bild zeigt:

Aufzeichnung 1	1	2	3	4	5	6	7	8	9	10
Aufzeichnung 2	101	102	103	104	105	106	107	108	109	110
Aufzeichnung 3	201	202	203	204	205	206	207	208	209	210
Aufzeichnung 4	301	302	303	304	305	306	307	308	309	310
Aufzeichnung 5	401	402	403	404	405	406	407	408	409	410

Das Programm liest dann diese Aufzeichnungen von der Datei und stellt sie auf dem Bildschirm dar. Das Programm wurde in BASIC 4.0 und BASIC < 3.0 geschrieben. Nachfolgend die Auflistung beider Programme mit den Namen SEQ.NUM.B4 und SEQ.NUM.B3:

SEQ.NUM. B4 (BASIC 4.0-Version)

```
10 REM  PROGRAMM ** SEQ.NUM.B4 **
20 DOPEN#1,"TESTDATEN" ,W
30 IF DS<>0 THEN PRINT DS$: STOP
40 REM  10 AUFZEICHNUNGEN A SCHREIBEN
50 FOR A=1 TO 10
60 REM  10 FELDER F JE AUFZEICHNUNG SCHREIBEN
70 FOR F=1 TO 10
80 PRINT#1,(A-1)*100+F
85 IF DS <>0 THEN PRINT DS$: STOP
90 NEXT  F
100 NEXT A
110 DCLOSE#1
200 REM  LESEN & DARSTELLEN DER AUFZEICHNUNGEN
210 DOPEN#1,"TESTDATEN"
215 IF DS<>0 THEN PRINT DS$ :STOP
220 FOR A=1 TO 10
230 PRINT "AUFZEICHNUNG";A;
240 REM  FELDER IN AUFZEICHNUNG A LESEN
250 FOR F=1 TO 10
260 INPUT#1,N
265 IF DS<>0 THEN PRINT DS$:STOP
270 PRINT N;
280 NEXT F
290 PRINT
300 NEXT A
310 DCLOSE#1
320 SCRATCH D0, "TESTDATEN"
330 STOP
```

SEQ.NUM. B3 (BASIC <3.0-Version)

```
10 REM  PROGRAMM ** SEQ.NUM.B<3" **
19 REM  OEFFNEN DES BEFEHLSKANALS
20 OPEN 15,8,15
21 INPUT#15,A$,B$,C$,D$
22 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
23 PRINT#15,"I0"
24 OPEN 1,8,2,"0:TESTDATEN,SEQ,W"
30 INPUT#15,A$,B$,C$,D$
31 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
40 REM  10 AUFZEICHNUNGEN A SCHREIBEN
50 FOR A=1 TO 10
60 REM  10 FELDER F JE AUFZEICHNUNG SCHREIBEN
70 FOR F=1 TO 10
80 PRINT#1,(A-1)*100+F
85 INPUT#15,A$,B$,C$,D$
86 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
90 NEXT F
100 NEXT A
```

```
110 CLOSE 1
120 CLOSE 15
200 REM  LESEN & DARSTELLEN DER AUFZEICHNUNGEN
209 REM  BEFEHLSKANAL OEFFNEN
210 OPEN 15,8,15
211 INPUT#15,A$,B$,C$,D$
212 IF VAL (A$)<>0 THEN PRINT A$,B$,C$,D$
213 OPEN 1,8,2,"0:TESTDATEN,SEQ"
215 INPUT#15,A$,B$,C$,D$
216 IF VAL (A$)<>0 THEN PRINT A$,B$,C$,D$
220 FOR A=1 TO 10
230 PRINT "AUFZEICHNUNG";A;
240 REM  FELDER IN AUFZEICHNUNG A LESEN
250 FOR F=1 TO 10
260 INPUT#1,N
265 INPUT#15,A$,B$,C$,D$
266 IF VAL (A$)<>0 THEN PRINT A$,B$,C$,D$
270 PRINT N;
280 NEXT F
290 PRINT
300 NEXT A
310 CLOSE 1
320 SCRATCH D0,"TESTDATEN"
330 CLOSE 15
340 STOP
```

Geben Sie das Programm in derjenigen Version ein, in der es auf Ihrem CBM-Computer ablaufen kann, prüfen Sie es auf Fehler, speichern Sie das Programm auf Bandkassette oder Diskette und lassen es dann ablaufen. Auf dem Bildschirm sollte dann folgende Darstellung erscheinen:

```
AUFZEICHNUNG 1   1   2   3   4   5   6   7   8   9   10
AUFZEICHNUNG 2   101   102   103   104   105   106   107   108   109   110
AUFZEICHNUNG 3   201   202   203   204   205   206   207   208   209   210
AUFZEICHNUNG 4   301   302   303   304   305   306   307   308   309   310
AUFZEICHNUNG 5   401   402   403   404   405   406   407   408   409   410
AUFZEICHNUNG 6   501   502   503   504   505   506   507   508   509   510
AUFZEICHNUNG 7   601   602   603   604   605   606   607   608   609   610
AUFZEICHNUNG 8   701   702   703   704   705   706   707   708   709   710
AUFZEICHNUNG 9   801   802   803   804   805   806   807   808   809   810
AUFZEICHNUNG 10   901   902   903   904   905   906   907   908   909   910
```

Lassen Sie uns jetzt den Programmaufbau untersuchen.

In Zeilen 10 bis 110 (120 in Version BASIC < 3.0) wird eine Sequentielle Datei für Daten eingerichtet, in die dann 10 Aufzeichnungen geschrieben werden. In Zeilen 200 bis 320 wird eine Aufzeichnung nach der anderen aus der Sequentiellen Datei gelesen und auf dem Bildschirm dargestellt.

Zunächst einen Blick auf das Öffnen und Schließen der Dateien.

In der Version BASIC 4.0 wird in Zeile 20 mit Hilfe der Anweisung DOPEN# die Sequentielle Daten-Datei TESTDATEN für einen Schreibvorgang geöffnet. DOPEN# verwendet die logische Dateinummer 1. In Zeile 110 erfolgt das Schließen dieser Datei, ehe sie zum Lesen erneut geöffnet wird (DOPEN#-Anweisung in Zeile 210). In Zeile 210 benutzt eine DOPEN#-Anweisung wieder die logische Dateinummer 1. Die Wiederbenutzung der gleichen logischen Dateinummer für die gleiche Datei ist jedoch nicht notwendig. Die logische Datei 1 wird endgültig in Zeile 310 geschlossen.

In der Version BASIC < 3.0 öffnet in Zeile 24 eine OPEN-Anweisung die Sequentielle Daten-Datei TESTDATEN < 3 für eine Schreiboperation. Die logische

Dateinummer ist 1 und die Sekundäradresse 2. Die Datei wird in Zeile 110 geschlossen, ehe sie in Zeile 213 mit der OPEN-Anweisung für das folgende Lesen wieder geöffnet wird. Die Datei wird endgültig in Zeile 310 geschlossen. Die BASIC < 3.0-Version öffnet außerdem einen Befehlskanal mit der OPEN-Anweisung in Zeile 20, wobei die Verwendung der logischen Dateinummer 15 freigestellt ist, während die Sekundäradresse 15 erforderlich ist. Es ist üblich, in BASIC < 3.0-Programmen für den Befehlskanal die logische Dateinummer 15 zu verwenden, da die Sekundäradresse diese Nummer mit dem Befehlskanal in Verbindung bringt. Der Befehlskanal wird in Zeile 120 geschlossen und in Zeile 210 erneut geöffnet und endgültig in Zeile 330 geschlossen. Das Schließen und erneute Öffnen des Befehlskanals ist nicht erforderlich, so daß die Zeilen 120 und 210 auch fehlen könnten. Durch Schließen und Wiederöffnen des Befehlskanals zerfällt jedoch das Programm in zwei Hälften, die wie getrennte Programm-Module unabhängig voneinander ablaufen können.

Beachten Sie, daß in Version BASIC < 3.0 die Diskette in Zeile 23 initiiert wird. Streng genommen sollte die Diskette bei Wiederöffnung des Befehlskanals in Zeile 210 erneut initiiert werden, damit die beiden Programmhälften als unabhängige Modulen behandelt werden können.

In beiden Programmversionen wird die Daten-Datei am Programmende gelöscht (Zeile 320). Würde das Löschen der Datei unterbleiben, dann wäre eine Wiederholung des Programmablaufs nicht möglich. Versuchen Sie, die Anweisung in Zeile 320 zu entfernen und das Programm erneut ablaufen zu lassen. Beim zweiten Programmablauf erhalten Sie die Fehlermeldung FILE ALREADY EXISTS, sobald die Datei für eine Schreiboperation geöffnet wird (in Zeile 20 der BASIC 4.0-Version und in Zeile 24 der BASIC < 3.0-Version).

Sie sollten kurzzeitig benutzte Dateien am Programmende löschen, wenn diese Dateien Daten enthalten, die nicht bewahrt werden müssen. Vorübergehend benutzte Dateien müssen gelöscht werden, da sie bei erneutem Programmablauf nicht wieder benutzt werden können.

Als nächstes ein Blick auf die Fehlerüberwachung der Disketten in beiden Programmversionen; diese Programmlogik fehlt in den meisten Programmen, die in Eile geschrieben wurden. Anweisungen in BASIC < 3.0 und BASIC 4.0 zum Lesen des Statusregisters und Feststellen von Fehlermeldungen des Floppy Disk-Speichers wurden weiter oben in diesem Kapitel beschrieben.

Das BASIC 4.0-Programm prüft in Zeilen 30, 85, 215 und 265 das Statusregister auf vorliegende Fehlermeldungen. Ist der Inhalt des Statusregisters nicht 0, dann wird zur Fehlerinterpretation die Variable DS$ auf dem Bildschirm dargestellt und der Programmlauf unterbrochen.

Das BASIC < 3.0-Programm verfügt über die gleiche Prüflogik, erkennt Fehlermeldungen jedoch durch Einlesen der Variablen A$, B$, C$ und D$. Ist der numerische Wert von A$ nicht 0, dann werden die vier Variablen auf dem Bildschirm dargestellt. Die Anweisungen dieser Prüflogik befinden sich in Zeilen 21 und 22, 30 und 31, 85 und 86, 210 und 211, 215 und 216, 265 und 266.

Die BASIC 4.0- und BASIC < 3.0-Programme benutzen identische Anweisungen, um Aufzeichnungen auf die Sequentielle Daten-Datei zu schreiben, Aufzeichnungen zu lesen und Daten auf dem Bildschirm darzustellen.

Von Zeilen 50 bis 100 werden Aufzeichnungen auf die Sequentielle Daten-Datei geschrieben. Die äußere FOR-NEXT-Schleife mit dem Index A zählt die Aufzeichnungen; die innere FOR-NEXT-Schleife mit dem Index F zählt die Felder innerhalb der Aufzeichnungen. Jedes Feld wird mit der PRINT#-Anweisung in Zeile 80 auf die Sequentielle Datei geschrieben. Da sich in der Parameterliste der PRINT#-Anweisung nur eine Variable befindet, wird zwangsläufig das Wagenrücklauf-Symbol geschrieben. Erinnern Sie sich: wenn nach Ersatz von PRINT# durch PRINT die Felder

in Form einer einzigen vertikalen Spalte auf dem Bildschirm dargestellt werden, dann werden sie auch richtig auf die Diskette geschrieben.

In Zeilen 220 bis 300 erfolgt das Lesen der Daten von der Sequentiellen Datei und ihre Darstellung auf dem Bildschirm. Die äußere FOR-NEXT-Schleife mit dem Index A dient dem Lesen der einzelnen Aufzeichnungen; die PRINT-Anweisung in Zeile 230 beginnt die Darstellung jeder Aufzeichnung mit der Aufzeichnungsnummer. Die innere FOR-NEXT-Schleife mit dem Index F liest jeweils ein Feld mit Hilfe der INPUT#-Anweisung in Zeile 260. Die Felder einer Aufzeichnung werden durch die PRINT-Anweisung in Zeile 270 auf einer Bildschirmzeile dargestellt. Nach erfolgter Darstellung einer Aufzeichnung löst die PRINT-Anweisung in Zeile 290 den Wagenrücklauf (Zeilensprung) aus.

Wir haben die Sequentielle Datei beschrieben, als ob sie aus 10 Aufzeichnungen mit je 10 Feldern besteht. Tatsächlich besteht die Sequentielle Datei auf der Oberfläche der Diskette aus 100 Feldern, die durch Wagenrücklauf-Symbole getrennt sind. Wenn Sie die gespeicherten Daten auf der Oberfläche der Diskette sehen könnten, würden Sie nichts finden, was Anfang und Ende einer Aufzeichnung erkennen ließe. Die Programmlogik muß durch Zählen der Felder in ihren FOR-NEXT-Schleifen die einzelnen Aufzeichnungen erkennen.

Um das Fehlen jedweder Dateistruktur auf der Diskette zu zeigen, können wir die zweite Programmhälfte derart ändern, daß beim Lesen der Datei zwölf Aufzeichnungen zu je 8 Feldern angenommen werden. Die Anweisungen in Zeilen 220 und 250 müssen hierfür wie folgt lauten:

```
220 FOR A=1 TO 12
250 FOR F=1 TO 8
```

Ein erneuter Programmlauf liefert Ihnen dann folgende Bildschirmdarstellung:

```
AUFZEICHNUNG 1   1   2   3   4   5   6   7   8
AUFZEICHNUNG 2   9   10   101'  102   103   104   105   106
AUFZEICHNUNG 3   107   108   109   110   201   202   203   204
AUFZEICHNUNG 4   205   206   207   208   209   210   301   302
AUFZEICHNUNG 5   303   304   305   306   307   308   309   310
AUFZEICHNUNG 6   401   402   403   404   405   406   407   408
AUFZEICHNUNG 7   409   410   501   502   503   504   505   506
AUFZEICHNUNG 8   507   508   509   510   601   602   603   604
AUFZEICHNUNG 9   605   606   607   608   609   610   701   702
AUFZEICHNUNG 10   703   704   705   706   707   708   709   710
AUFZEICHNUNG 11   801   802   803   804   805   806   807   808
AUFZEICHNUNG 12   809   810   901   902   903   904   905   906
```

Die Gruppierung von Feldern zu Aufzeichnungen beim Lesen der Datei geschieht ohne Rücksicht auf die Gruppierungen, die beim Schreiben der Datei verwendet wurden.

Wenn mit einer einzelnen PRINT#-Anweisung zwei oder mehr numerische Variable in eine Datei für Daten geschrieben werden, dann muß das Wagenrücklauf-Symbol durch die Funktion CHR$(13) dargestellt werden. Nehmen Sie Zeile 80 unseres Programms: Wenn wir neben dem berechneten Ausdruck auch A und F ausgeben wollen, dann müßte die PRINT#-Anweisung wie folgt aussehen:

```
80 PRINT#1,A,CHR$(13),F,CHR$13),(A-1)*100+F
```

Für gewöhnlich stellt man das Wagenrücklauf-Symbol durch eine Zeichenketten-Variable dar, die dann in der PRINT#-Anweisung wie folgt benutzt wird:

```
15 C$=CHR$(13)
.
.
80 PRINT#1,A,C$,F,C$,(A-1)*100+F
```

Jede Aufzeichnung besteht jetzt aus 30 Zahlen (A und F sind hinzugekommen) und nicht aus 10. Daher müssen in der zweiten Programmhälfte für jede Aufzeichnung 30 Zahlen gelesen und auf dem Bildschirm dargestellt werden. Ein einfacher, aber nicht sehr eleganter Weg zur Darstellung von 30 Zahlen besteht in der Änderung der FOR-NEXT-Schleife in Zeile 250 durch Heraufsetzen der oberen Grenze des Index auf 30:

```
250 FOR F=1 TO 30
```

Wenn Sie diese Änderungen in Ihr Programm übernehmen, können Sie bestätigt finden, daß bei jeder Ausführung der PRINT#Anweisung in Zeile 80 drei Zahlen in die Datei geschrieben wurden.

SCHREIBEN VON ZEICHENKETTEN IN EINE SEQUENTIELLE DATEI

Zeichenketten-Variable können durch Komma- oder Wagenrücklauf-Symbole getrennt werden. Für das Speichern von Zeichenketten-Variablen in Sequentiellen Dateien ist jedoch das Komma als Trennsymbol nicht nützlich. Wir werden daher alle Zeichenketten in Sequentiellen Dateien durch Wagenrücklauf-Symbole trennen.

Zeichenketten und numerische Daten können mit der gleichen Programmlogik in eine Sequentielle Datei geschrieben werden.

Wir werden ein einfaches Versandlisten-Programm schreiben, um das Speichern von Zeichenketten in einer Sequentiellen Datei zu zeigen. Das Programm wurde in den Versionen BASIC 4.0 und BASIC < 3.0 geschrieben und ist zusammen mit dem Ergebnis eines Programmlaufs nachfolgend aufgelistet:

SEQ.ADR.LISTE. B4 (BASIC 4.0-Version)

```
10 REM  PROGRAMM ** SEQ.ADR.LISTE.B4 **
20 REM  ADRESSLISTEN-PROGRAMM ZUR DEMONSTRATION DER
21 REM  SPEICHERUNG VON ZEICHENKETTEN AUF DISKETTEN
30 DATA " NAME    : "," BRANCHE: "," STRASSE: "
31 DATA " STADT   : "," LAND    : "
40 DOPEN#1, "SEQ.ADR.LISTE",W
50 IF DS<>0 THEN PRINT DS$: STOP
60 PRINT"■ BITTE ADRESSE EINGEBEN :■■"
70 FOR I=1 TO 5
80 READ F$
90 PRINT F$;:INPUT AD$(I)
100 NEXT I
110 RESTORE
120 PRINT "■SPEICHERN? J EINGEBEN ** AENDERN? N EINGEBEN"
130 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 130
135 PRINT Y$
140 IF Y$="N" THEN 60
150 REM  ADRESSE AUF SEQUENTIELLE DATEI SCHREIBEN
160 FOR I=1 TO 5
170 PRINT#1,AD$(I)
180 NEXT I
190 PRINT "WEITERE ADRESS-EINGABEN? J ODER N EINGEBEN"
200 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 200
205 PRINTY$
210 IF Y$="J" THEN 60
```

```
220 DCLOSE#1
300 REM  ADRESSEN EINZELN WIEDERGEBEN
310 DOPEN#1, "SEQ.ADR.LISTE"
330 IF DS<>0 THEN PRINT DS$: STOP
340 REM  BILDSCHIRM LOESCHEN & ADRESSE DARSTELLEN
350 PRINT"■■■"
360 RESTORE
370 FOR I=1 TO 5
380 READ F$: PRINT F$;
390 INPUT#1,AD$
400 IF DS<>0 THEN PRINT DS$: STOP
410 PRINT AD$
420 NEXT I
430 PRINT "■WEITERE ADRESSE DARSTELLEN? J ODER N EINGEBEN"
440 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 440
450 IF Y$="J" THEN 350
460 DCLOSE#1
470 SCRATCH D0,"SEQ.ADR.LISTE"
480 STOP
```

SEQ.ADR.LISTE. B3 (BASIC <3.0-Version)

```
10 REM  PROGRAM ** SEQ.ADR.LISTE.B<3 **
20 REM  ADRESSLISTEN-PROGRAMM ZUR DEMONSTRATION DER
21 REM  SPEICHERUNG VON ZEICHENKETTEN AUF DISKETTEN
30 DATA " NAME    : "," BRANCHE: "," STRASSE: "
31 DATA " STADT   : "," LAND    : "
39 REM  OEFFNEN DES BEFEHLSKANALS
40 OPEN 15,8,15
41 INPUT#15,A$,B$,C$,D$
42 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
43 PRINT#15,"I0"
44 OPEN 1,8,2,"0:ADR.LISTE.B<3,W"
50 INPUT#15,A$,B$,C$,D$
51 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
60 PRINT "■ BITTE ADRESSE EINGEBEN :■■"
70 FOR I=1 TO 5
80 READ F$
90 PRINT F$;:INPUT AD$(I)
100 NEXT I
110 RESTORE
120 PRINT "■SPEICHERN? J EINGEBEN ** AENDERN? N EINGEBEN"
130 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 130
135 PRINT Y$
140 IF Y$="N" THEN 60
150 REM  ADRESSE AUF SEQUENTIELLE DATEI SCHREIBEN
160 FOR I=1 TO 5
170 PRINT#1,AD$(I)
180 NEXT I
190 PRINT "WEITERE ADRESS-EINGABEN? J ODER N EINGEBEN"
200 GET Y$:IF Y$<>"J" AND Y$<>"N" THEN 200
205 PRINTY$
210 IF Y$="J" THEN 60
220 CLOSE 1
300 REM  ADRESSEN EINZELN WIEDERGEBEN
310 OPEN 1,8,2,"0:ADR.LISTE.B<3"
320 REM STATUS-PRUEFUNG DER DISKETTE
321 INPUT#15,A$,B$,C$,D$
330 IF VAL(A$)<>0 THEN PRINT A$,B$,C$,D$
340 REM  BILDSCHIRM LOESCHEN UND ADRESSE DARSTELLEN
350 PRINT "■■■"
360 RESTORE
```

```
370 FOR I=1 TO 5
380 READ F$:PRINT F$;
390 INPUT#1,AD$
400 INPUT#15,A$,B$,C$,D$
401 IF VAL (A$)<>0 THEN PRINT A$,B$,C$,D$
410 PRINT AD$
420 NEXT I
430 PRINT "■WEITERE ADRESSE DARSTELLEN? J ODER N EINGEBEN"
440 GET Y$:IF Y$<>"J" AND Y$<>"N" THEN 440
450 IF Y$="J" THEN 350
460 CLOSE 1
470 SCRATCH D0,"ADR.LISTE.B<3"
480 STOP
```

```
BITTE ADRESSE EINGEBEN :

 NAME    : ? FA. VAN LEEUWEN
 BRANCHE: ? INDUSTRIEUNTERNEHMUNG
 STRASSE: ? WESTERBAENSTR.50
 STADT   : ? 2513 GH DEN HAAG
 LAND    : ? NL

SPEICHERN? J EINGEBEN ** AENDERN? N EINGEBEN
J
WEITERE ADRESSEINGABEN? J ODER N EINGEBEN
N

 NAME    :  FA. VAN LEEUWEN
 BRANCHE:  INDUSTRIEUNTERNEHMUNG
 STRASSE:  WESTERBAENSTR.50
 STADT   :  2513 GH DEN HAAG
 LAND    :  NL

WEITERE ADRESSE DARSTELLEN? J ODER N EINGEBEN
J



 NAME    :  FA. WOLFGANG RIEGER
 BRANCHE:  KANALREINIGUNG
 STRASSE:  RICHARD-STOPF-STR.65
 STADT   :  8032 GRAEFELFING
 LAND    :  BRD
```

Lassen Sie uns den Programmaufbau untersuchen.

Die Eingabe von Namen und Adressen über das Tastenfeld und ihre Speicherung auf einer sequentiellen Daten-Datei erfolgen in Zeilen 40-220. Das Lesen von Namen und Adressen aus der Datei und ihre Darstellung auf dem Bildschirm erfolgen in Zeilen 300 bis 460.

In der BASIC 4.0-Version trägt die Sequentielle Datei den Namen SEQ.ADR. LISTE. In Zeile 40 wird diese Datei für eine Schreiboperation geöffnet und in Zeile 220 geschlossen. Die Datei wird erneut in Zeile 310 für eine Leseoperation geöffnet und in Zeile 460 wieder geschlossen.

In der Version BASIC < 3.0 trägt die Sequentielle Datei den Namen ADR.LISTE.B<3. Diese Datei wird in Zeile 44 zum Schreiben geöffnet und in Zeile 220 geschlossen. Die Datei wird zum Lesen in Zeile 310 erneut geöffnet und endgültig in Zeile 460 geschlossen.

Beide Programme löschen die Sequentielle Datei in Zeile 470, so daß der Programmlauf wiederholt werden kann. In einem praktischen Versandadressen-Pro-

gramm würde die Datei nicht gelöscht werden; die Versandadressen müssen erhalten bleiben. Stattdessen müßten weitere Namen und Adressen an die Datei angefügt werden. Im nächsten Kapitel wird das Anfügen von Daten an sequentielle Dateien beschrieben.

Fehlermeldungen des Floppy Disk-Speichers werden in der BASIC 4.0-Version des Programms mit den Anweisungen in Zeilen 50, 330 und 400 festgestellt. In der BASIC < 3.0-Version erfolgt dieser Fehlertest in Zeilen 41 und 42, 50 und 51, 321 und 330, 400 und 401. Fehlererkennung und Fehlerinterpretation wurden weiter oben in diesem Kapitel beschrieben.

Beachten Sie, daß die BASIC < 3.0-Version am Programmanfang in Zeile 40 einen Befehlskanal öffnet. Das Schließen des Kanals wurde der STOP-Anweisung in Zeile 480 übertragen; das ist zwar nicht gute Programmierpraxis, aber zulässig.

Die Versionen BASIC 4.0 und BASIC < 3.0 nutzen identische Anweisungen, um Daten vom Tastenfeld zu lesen, Daten auf die Sequentielle Datei zu schreiben, Daten von der Sequentiellen Datei zu lesen und Daten auf dem Bildschirm darzustellen.

Das Einlesen der Namen und Adressen vom Tastenfeld erfolgt in Zeilen 60 bis 140. Die FOR-NEXT-Schleife in Zeilen 70 bis 100 bildet fünf Felder aus den Angaben von Name und Adresse, die in Zeile 30 von der DATA-Anweisung gespeichert und in Zeile 80 der Reihe nach mit einer READ-Anweisung in die Variable F$ gelesen werden. Die fünf Felder für Name und Adresse werden in Zeile 90 in einer Tabelle für Zeichenketten, AD$(I), gespeichert. Die RESTORE-Anweisung in Zeile 110 setzt den Lesezeiger auf die erste Zeichenketten-Variable in der DATA-Anweisung in Zeile 30 zurück.

Ein Dialog mit dem Benutzer zur Wiederholung oder Bestätigung der Namen- und Adresseingaben befindet sich auf Zeilen 120 bis 140. Diese Art von Dialog wurde häufig in Kapitel 5 beschrieben. Die Programmlogik zur Fehlererkennung ist sehr einfach gehalten, da wir eigentlich das Arbeiten mit Dateien zeigen wollen.

Name und Adresse werden durch die FOR-NEXT-Schleife in Zeilen 160 bis 180 auf die Sequentielle Datei geschrieben. Da bei jeder Ausführung der PRINT#-Anweisung in Zeile 170 nur eine Zeichenkette ausgegeben wird, wird jede Variable zwangsweise mit einem Wagenrücklauf-Symbol abgeschlossen. Die Anweisungen in Zeilen 160 bis 180 könnten auch durch folgende zwei Anweisungen ersetzt werden:

```
160 C$=CHR$(13)
170 PRINT#1,AD$(1),C$,AD$(2),C$,AD$(3),C$,AD$(4),C$,AD$(5)
```

Die folgende INPUT#-Anweisung kann wahlweise zum Lesen der Daten von der Datei benutzt werden:

```
200 INPUT#1,AD$(1),AD$(2),AD$(3),AD$(4),AD$(5)
```

In Zeilen 10 bis 210 kann der Benutzer entscheiden, ob eine weitere Eingabe von Name und Adresse erfolgen oder das Programm zur Datendarstellung übergehen soll.

Das Lesen der fünf Felder für Name und Adresse und ihre Darstellung auf dem Bildschirm erfolgt in der FOR-NEXT-Schleife in Zeilen 370 bis 420. Jedem Feld wird seine Bezeichnung (Name, Straße usw.) vorangestellt. Diese Bezeichnungen werden in Zeile 380 mit einer READ-Anweisung aus der DATA-Anweisung in Zeile 30 gelesen, der Variablen F$ zugewiesen und mit einer PRINT-Anweisung auf dem Bildschirm dargestellt. Danach liest die INPUT#-Anweisung in Zeile 390 das zugehörige Feld von der Sequentiellen Datei und stellt es mit der PRINT-Anweisung in Zeile 410 auf dem Bildschirm dar.

Ein Dialog mit dem Benutzer in Zeilen 430 bis 450 entscheidet, ob eine weitere Namen- und Adressdarstellung erfolgen oder der Programmlauf beendet werden soll.

Beachten Sie, daß das Programm beim Erreichen des Dateiendes den Benutzer nicht daran hindert, nach einer weiteren Namen- und Adressdarstellung zu fragen. Dieser Mangel kann mit folgenden Anweisungen in einer neuen Zeile 405 beseitigt werden:

```
405 IF DS=64 THEN PRINT "DATEI-ENDE": I=5: GOTO 420
```

SEQUENTIELLE DATEIEN FÜR GEMISCHTE DATEN

Numerische Daten und Zeichenketten können ohne besondere Programmlogik in die gleiche Sequentielle Datei geschrieben werden. Die Programmlogik muß sich jedoch merken, in welcher Reihenfolge die verschiedenen Variablentypen auftraten. Wenn eine Anweisung beim Lesen eines Feldes aus einer Sequentiellen Daten-Datei einen Variablennamen des falschen Typs verwendet, dann erfolgt eine Fehlermeldung.

Hier das Beispiel für eine Anweisung, die zwei numerische Variable und drei Zeichenketten-Variable in eine Sequentielle Daten-Datei schreibt:

```
10 DOPEN#1,"DATA",W
20 C$=CHR$(13)
30 PRINT#1,P$,C$,X,C$,Q$,C$,Y,C$,R$
```

Diese fünf Variablen können mit folgender INPUT#-Anweisung unter richtiger Beachtung ihres Typs von der Datei zurückgelesen werden:

```
100 INPUT#1,A$(1),X(1),A$(2),X(2),A$(3)
```

Die folgende INPUT#-Anweisung wird jedoch nicht richtig ausgeführt, da der Typ Ihrer Variablen in der Parameter-Liste nicht mit dem Typ der Variablen in der Datei-Aufzeichnung übereinstimmt:

```
100 INPUT#1,A$(1),A$(2),A$(3),X(1),X(2)
```

ANFÜGEN VON DATEN AN SEQUENTIELLE DATEIEN

BASIC 4.0 ermöglicht das Anfügen von Daten an eine bestehende Sequentielle Datei mit Hilfe der Anweisungen APPEND# und CONCAT. Die APPEND#-Anweisung schreibt Felder an das Ende einer bestehenden Datei; die CONCAT-Anweisung verkettet zwei Dateien.

ANFÜGEN VON DATEN MIT APPEND# IN BASIC 4.0

Zur Darstellung der APPEND#-Anweisung wandeln wir das Programm SEQ.NUM.B4 ab. Das abgewandelte Programm trägt den Namen SEQ.NUM.APPEND# und ist nachfolgend aufgelistet, wobei die Programmänderungen markiert wurden:

SEQ.NUM.APPEND #

```
10 REM  PROGRAMM ** SEQ.NUM.APPEND **
20 DOPEN#1,"TESTDATEN" ,W
30 IF DS<>0 THEN PRINT DS$: STOP
35 FOR I=1 TO 3
40 REM  10 AUFZEICHNUNGEN A SCHREIBEN
50 FOR A=1 TO 10
60 REM  10 FELDER F JE AUFZEICHNUNG SCHREIBEN
70 FOR F=1 TO 10
80 PRINT#1,(A-1)*100+F*I
```

```
85 IF DS <>0 THEN PRINT DS$: STOP
90 NEXT  F
100 NEXT A
110 DCLOSE#1
200 REM  LESEN & DARSTELLEN DER AUFZEICHNUNGEN
210 DOPEN#1,"TESTDATEN"
215 IF DS<>0 THEN PRINT DS $:STOP
220 FOR A=1 TO 10*J
230 PRINT "AUFZEICHNUNG";A;
240 REM
250 FOR F=1 TO 10
260 INPUT#1,N
265 IF DS<>0 THEN PRINT DS$:STOP
270 PRINT N;
280 NEXT F
290 PRINT
300 NEXT A
310 DCLOSE#1
315 APPEND#1,"TESTDATEN"
316 NEXT J
320 SCRATCH D0, "TESTDATEN"
330 STOP
```

Das Programm SEQ.NUM.APPEND# entspricht einem dreifachen Programmlauf des Programms SEQ.NUM.B4. Beim ersten Programmlauf werden 100 numerische Felder in die Datei TESTDATA geschrieben. Bei jedem folgenden Programmlauf werden weitere 100 Felder an die Sequentielle Datei TESTDATA angefügt. Nach dem 2. Programmlauf sind in TESTDATA 200 Zahlen gespeichert und nach dem 3. Programmlauf 300 Zahlen.

Die 3 Programmläufe werden von einer FOR-NEXT-Schleife mit dem Index J gesteuert. Die FOR-Anweisung befindet sich in Zeile 35, die NEXT-Anweisung in Zeile 316.

Zum Erkennen angefügter Zahlen wird der Zählerstand F für Felder mit dem Zählerstand J für Programmläufe multipliziert (Zeile 80). In Zeile 220 wird die obere Grenze für den Zähler A der Aufzeichnungen auf den Wert 10 mal J begrenzt, da die Zahl der Aufzeichnungen sich nach jedem Programmlauf um 10 erhöht.

Die APPEND-Anweisung kann nicht für Dateien verwendet werden, die nicht existieren. Sie können daher nicht einfach die DOPEN#-Anweisung in Zeile 20 durch eine APPEND#-Anweisung ersetzen und die Datei TESTDATA innerhalb der FOR-NEXT-Schleife mit dem Index J öffnen. Mit der DOPEN#-Anweisung in Zeile 20 wird die Sequentielle Datei TESTDATA eingerichtet und für eine Schreiboperation geöffnet. Bei der ersten Ausführung der Anweisungen in Zeilen 40 bis 315 werden 10 Aufzeichnungen auf die Datei TESTDATA geschrieben; diese 10 Aufzeichnungen werden anschließend gelesen und auf dem Bildschirm dargestellt. Am Ende des ersten Programmlaufs wird die Datei TESTDATA mit APPEND# in Zeile 315 erneut geöffnet. Hierdurch ist ein zweiter Programmlauf mit den Anweisungen in Zeilen 40 bis 315 möglich. 10 weitere Aufzeichnungen werden an TESTDATA angefügt. Ähnlich verläuft der dritte Programmlauf mit den Anweisungen in Zeilen 40 bis 315, an dessen Ende weitere 10 Aufzeichnungen an TESTDATA angefügt sind, so daß sich insgesamt 30 Aufzeichnungen ergeben.

Lassen Sie jetzt das Programm ablaufen. Beim ersten Programmlauf zeigt sich die gleiche Bildschirmdarstellung wie beim Programm SEQ.NUM.B4. Nach einer Pause werden im zweiten Programmlauf 20 Aufzeichnungen dargestellt; die letzten 10 Aufzeichnungen können Sie daran erkennen, daß die letzte Stelle jeder Zahl verdoppelt ist. Nach einer weiteren Pause erscheint der dritte Satz von 10 Aufzeichnungen auf dem Bildschirm. Die Darstellung des ersten, zweiten und dritten Satzes von je

10 Aufzeichnungen können Sie jeweils an der letzten Stelle jeder Zahl erkennen, die beim zweiten Satz von Aufzeichnungen verdoppelt und beim dritten Satz von Aufzeichnungen verdreifacht ist.

VERKETTUNG SEQUENTIELLER DATEIEN IN BASIC 4.0

In BASIC 4.0 mit dem Disketten-Betriebssystem DOS 2.0 ermöglicht die CONCAT-Anweisung eine Verkettung von Dateien, wie das unten aufgelistete Programm SEQ.TEST.CONCAT zeigt:

SEQ.TEST.CONCAT

```
7 REM  PROGRAMM ** CONCAT TEST **
8 REM  VERKETTEN VON DATEIEN MIT "CONCAT"
9 REM  IN JEDE DATEI 20 ZAHLEN SCHREIBEN:
10 DOPEN#1,"DATEN 1",W
20 DOPEN#2,"DATEN 2",W
30 FOR I=1 TO 20
40 PRINT#1,I
50 PRINT#2,I+10
60 NEXT I
70 DCLOSE
71 REM  INHALT JEDER DATEI DARSTELLEN:
80 DOPEN#1,"DATEN 1"
90 DOPEN#2,"DATEN 2"
100 PRINT"■"
110 FOR I=1 TO 20
120 INPUT#1,X :PRINT X;
130 NEXT
140 PRINT
150 FOR I=1 TO 20
160 INPUT#2,X :PRINT X;
170 NEXT
180 PRINT
190 DCLOSE
191 REM  DATEI 2 AN DATEI 1 ANFUEGEN:
200 CONCAT "DATEN 2" TO "DATEN 1"
201 REM  DATEI 1 MIT ANGEFUEGTER DATEI 2 DARSTELLEN
210 DOPEN#1,"DATEN 1"
220 FOR I=1 TO 40
230 INPUT#1,X :PRINT X;
240 NEXT
250 PRINT
260 DCLOSE
270 STOP
```

Dieses sehr einfache Programm schreibt 20 Zahlen in die Sequentiellen Dateien DATEN 1 und DATEN 2 und verkettet sie anschließend. Die Inhalte von DATEN 1 und DATEN 2 werden zunächst getrennt dargestellt, dann die Verkettung der Inhalte in DATEN 1.

Beide Sequentiellen Dateien DATEN 1 und DATEN 2 werden in Zeilen 10 und 20 geöffnet. Eine FOR-NEXT-Schleife in Zeilen 30-60 schreibt 20 numerische Felder in jede der beiden Dateien. Dabei werden die Zahlen 1-20 in DATEN 1 geschrieben, die Zahlen 11-31 in DATEN 2, so daß beide Zahlenfolgen leicht voneinander zu unterscheiden sind.

Beide Dateien werden mit einer einzigen DCLOSE-Anweisung in Zeile 70 geschlossen, um dann mit DOPEN#-Anweisungen in Zeilen 80 und 90 für einen Lesevorgang wieder geöffnet zu werden. Die FOR-NEXT-Schleifen in Zeilen 110 - 130 und 150 - 170 stellen den Inhalt beider Dateien auf dem Bildschirm dar. PRINT-Anweisungen in Zeilen 140 und 10 lösen jeweils einen Wagenrücklauf bzw. Zeilensprung

aus. Die Dateien DATEN 1 und DATEN 2 werden in Zeile 190 wieder geschlossen. Die Verkettung von DATEN 1 mit DATEN 2 erfolgt in Zeile 200 mit einer CONCAT-Anweisung. Danach wird DATEN 1 geöffnet, um seinen Inhalt mit einer FOR-NEXT-Schleife in Zeilen 220 bis 240 auf dem Bildschirm darzustellen. DATEN 1 wird dann in Zeile 260 geschlossen.

Beachten Sie, daß DATEN 1 und DATEN 2 am Programmende nicht gelöscht werden. Vor Wiederholung des Programmlaufs müssen Sie diese Dateien in direktem Dialog mit einer SCRATCH-Anweisung oder mit folgender Anweisung am Programmende löschen:

```
265 SCRATCH "DATEN 1" : SCRATCH "DATEN 2"
```

Die CONCAT-Anweisung kann leicht falsch benutzt werden und Schwierigkeiten verursachen. Beide verketteten Dateien müssen Daten enthalten und geschlossen sein, ehe eine CONCAT-Anweisung ausgeführt wird.

Werden leere Datein verkettet, bleibt der Computer hängen. Sie müssen dann den Computer ausschalten und wieder einschalten.

Wenn Sie versuchen, offene oder unvorschriftsmäßig geschlossene Dateien zu verketten, dann verkettet der Computer möglicherweise eine Datei mit dem Inhaltsverzeichnis der Diskette. Wenn dies geschieht, dann können Sie auch lange nach Ausführung der CONCAT-Anweisung das FLOPPY-DISK-Gerät noch in Betrieb sehen. Durch Drücken der STOP-Taste am Bedienfeld beenden Sie diesen Zustand. Stellen Sie dann das Inhaltsverzeichnis auf dem Bildschirm dar, können Sie unter den gültigen Dateinamen viel sinnlose Information finden. Um diese sinnlose Information zu entfernen, führen Sie in direktem Dialog mit dem Computer eine COLLECT-Anweisung aus.

ANFÜGEN VON DATEN AN SEQUENTIELLE DATEIEN IN BASIC<3.0

Zum Anfügen von Daten an eine bestehende Sequentielle Datei benötigen Sie in BASIC <3.0 zwei Sequentielle Dateien, die wir willkürlich mit DATA 1 und DATA 2 benennen. Um an DATA 1 Daten anzufügen, müssen wir die neue Datei DATA 2 in folgenden Schritten schaffen:

1. Löschen von DATA 2, falls es zuvor existierte.
2. Öffnen von DATA 2 für einen Schreibvorgang.
3. Öffnen von DATA 1 für einen Lesevorgang.
4. Sequentielles Lesen der Aufzeichnungen in DATA 1 und ihr sequentielles Einschreiben in DATA 2.
5. Schreiben neuer Aufzeichnungen in DATA 2, sobald das Dateiende von DATA 1 erkannt ist.
6. Schließen von DATA 1.
7. Löschen von DATA 1.
8. Ändern des Namens DATA 2 in DATA 1.

ERKENNEN EINES DATEIENDES

Das Ende einer Datei wird im Statusregister ST mit dem Zahlenwert 64 angezeigt. Die folgende Anweisung unterbricht den Programmlauf, sobald das Dateiende erkannt wurde:

```
200 IF ST=64 THEN PRINT "DATEI-ENDE" : STOP
```

Relative Dateien sind nur mit BASIC 4.0 möglich.

Eine geöffnete Relative Datei kann beschrieben oder gelesen werden. Das Lesen einer leeren Relativen Datei ist jedoch nicht möglich; ehe Sie etwas in die Datei geschrieben haben, können Sie nichts von ihr auslesen.

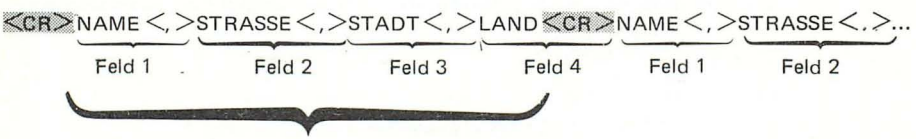
TRENNSYMBOLE ZWISCHEN FELDERN

Die Trennsymbole Komma und RETURN spielen in Relativen Dateien unterschiedliche Rollen. Eine Relative Datei baut sich aus Aufzeichnungen fester Länge auf, die sich voneinander mit dem Trennsymbol RETURN abgrenzen. Die Länge einer Aufzeichnung wird durch eine DOPEN#-Anweisung festgelegt und ist gleich der Zahl der Symbole (Bytes) zwischen zwei RETURN-Symbolen. Setzt sich eine Aufzeichnung aus Zeichenketten-Feldern zusammen, dann dient das Komma als Trennsymbol zwischen den Feldern. Ist jedoch die Länge eines Feldes gleich der Länge einer Aufzeichnung, dann besteht eine Relative Datei nur aus Feldern, die mit dem Symbol RETURN getrennt sind. Beachten Sie: PRINT#-Anweisungen in BASIC 4.0 übertragen nicht automatisch am Zeilenende das Trennsymbol RETURN, wenn die logische Datei eine Zahl 127 oder geringer ist.

WAHL DER LÄNGE EINER AUFZEICHNUNG

Numerische Felder müssen mit dem Trennsymbol RETURN abgeschlossen werden, so daß in Relativen Dateien für numerische Daten die Länge der Aufzeichnungen zugleich auch die Länge der numerischen Felder festlegt. Die Zahl nach dem Parameter L in einer DOPEN#-Anweisung für Relative Dateien legt die Zahl der Symbole (Bytes) fest, die für jedes numerische Feld in einer Relativen Datei reserviert sind.

Zeichenketten-Felder können dagegen sowohl durch ein Komma, als auch durch RETURN abgeschlossen werden, so daß eine Aufzeichnung eine Reihe von Zeichenketten-Feldern enthalten kann. Ein Beispiel: Eine Adreßangabe könnte aus folgenden vier Feldern bestehen:



Umfang einer Aufzeichnung in einer Relativen Datei

Die Länge einer Aufzeichnung, wie sie in einer DOPEN#-Anweisung festgelegt würde, gilt für jeweils vier Felder einer Adreßangabe. Die Zusammenfassung von Feldern zufällig schwankender Länge in einer Aufzeichnung ermöglicht kürzere Aufzeichnungslängen, wie folgendes Beispiel zeigt. Eine Statistik über fünf Adreßangaben zeigte folgende Schwankungen der Feldlängen:

	NAME	STRASSE	STADT	LAND	
	Feld 1	Feld 2	Feld 3	Feld 4	Total
Adresse 1	9	22	16	2	49
Adresse 2	13	12	8	3	36
Adresse 3	12	11	12	3	38
Adresse 4	17	13	11	3	44
Adresse 5	10	20	13	2	45
etc.					

Wenn alle vier Felder in einer einzigen Aufzeichnung gespeichert werden, könnte eine Aufzeichnungslänge von 50 Symbolen (Bytes) genügen.

Wenn dagegen jedes Feld mit einem RETURN abgeschlossen wird, dann gilt die Aufzeichnungslänge gemäß der Angabe in DOPEN# für jedes Feld der Adreßangabe. Die Aufzeichnungslänge müßte dann lang genug sein, um auch das längste zu erwartende Feld der Adreßangabe aufnehmen zu können. Um sicher zu sein, würden wir vielleicht eine Feldlänge von 25 Zeichen wählen. Dies wäre dann auch die Länge jeder Aufzeichnung. Jede Adreßangabe belegt jetzt einen Speicherplatz von 100 Zeichen, entsprechend vier Feldern zu je 25 Zeichen. Die Zusammenfassung von Feldern in einer Aufzeichnung hätte dagegen nur halb so viel Speicherplatz erfordert.

LESEN VON AUFZEICHNUNGEN IN EINER RELATIVEN DATEI

Aufzeichnungen und Felder in Relativen Dateien können mit den Anweisungen INPUT# und GET# gelesen werden. Liegen zwischen zwei RETURN-Symbolen Zeichenketten, die durch Kommas getrennt sind, dann liest eine INPUT#-Anweisung jeweils alle Zeichenketten, die sie zwischen zwei RETURN-Symbolen vorfindet. Wir werden hierfür auf den folgenden Seiten Programmbeispiele geben.

Besteht eine Relative Datei aus numerischen Feldern und Feldern für Zeichenketten, dann wird die Wahl einer geeigneten Aufzeichnungslänge schwieriger. Da in einer Aufzeichnung jeweils eine Reihe von Zeichenketten-Feldern, aber nur ein numerisches Feld gespeichert werden darf, kann sich das Problem ergeben, daß die für Zeichenketten gewählte Aufzeichnungslänge zu groß ist für numerische Felder. Hierdurch geht Speicherplatz auf der Diskette verloren. Es gibt zwei Lösungen zu diesem Problem:

1. Die Aufzeichnungslänge stimmt mit der Länge numerischer Felder überein. In diesem Falle wird je Aufzeichnung nur ein Zeichenketten-Feld gespeichert, wobei lange Zeichenketten in kürzere Abschnitte zu teilen sind.
2. Die Aufzeichnungslänge stimmt mit der Länge der Zeichenketten-Felder überein. In diesem Fall wandelt man numerische Felder mit Hilfe der Funktion STR$ in Zeichenketten-Felder um und speichert sie dann wie Zeichenketten.

SCHREIBEN NUMERISCHER DATEN IN RELATIVE DATEIEN

Um das Schreiben von Zahlen in Relative Dateien zu erkunden, haben wir das Programm SEQ.NUM.B4 abgewandelt und das nachfolgend aufgelistete Programm REL.NUM.B4 gebildet:

REL.NUM. B4 (BASIC 4.0-Version)

```
10 REM  PROGRAMM ** REL.NUM.B4 **
20 DOPEN#1,"TESTDATEN" ,L10
30 IF DS<>0 THEN PRINT DS$: STOP
40 REM  10 AUFZEICHNUNGEN A SCHREIBEN
50 FOR A=1 TO 10
60 REM  10 FELDER F JE AUFZEICHNUNG SCHREIBEN
70 FOR F=1 TO 10
80 PRINT#1,(A-1)*100+F
85 IF DS <>0 THEN PRINT DS$: STOP
90 NEXT  F
100 NEXT A
110 DCLOSE#1
200 REM  LESEN & DARSTELLEN DER AUFZEICHNUNGEN
210 DOPEN#1,"TESTDATEN",L10
215 IF DS<>0 THEN PRINT DS $:STOP
220 FOR A=1 TO 10
230 PRINT "AUFZEICHNUNG";A;
250 FOR F=1 TO 10
260 INPUT#1,N
265 IF DS<>0 THEN PRINT DS$:STOP
270 PRINT N;
280 NEXT F
290 PRINT
300 NEXT A
310 DCLOSE#1
320 SCRATCH D0, "TESTDATEN"
330 STOP
```

Die Programmänderungen wurden durch Schattierung kenntlich gemacht. Laden Sie das Programm SEQ.NUM.B4 in den Arbeitsspeicher des Computers und ändern Sie es in Zeilen 10, 20 und 210, um das Programm REL.NUM.B4 zu bilden. Lassen Sie das Programm mit RUN ablaufen und sehen Sie, wie es die gleiche Bildschirmdarstellung erzeugt wie das Programm SEQ.NUM.B4. Speichern Sie danach das Programm REL.NUM.B4 auf Diskette oder Kassette.

AUFZEICHNUNGSLÄNGE NUMERISCHER DATEN

Beachten Sie die kurze Aufzeichnungslänge von 10 Zeichen (Bytes), die in den DOPEN#-Anweisungen in Zeilen 20 und 210 festgelegt wurde. Da eine Aufzeichnung jeweils n ur ein numerisches Feld speichern darf, ist für praktische Fälle eine Länge von 10 Zeichen (Bytes) ausreichend.

Eigentlich besteht kein Anlaß, die Relative Datei TESTDATEN in Zeile 110 zu schließen, um sie dann in Zeile 210 wieder zu öffnen. Wir haben es getan, um das Programm in zwei Module zu trennen und das Wechselspiel zwischen beiden Modulen untersuchen zu können.

Ändern Sie als nächstes die Aufzeichnungslänge in der DOPEN#-Anweisung in Zeile 210 von L10 in L8. Das Programm läuft jetzt nicht ab; statt dessen erscheint folgende Fehlermeldung:

```
50,RECORD NOT PRESENT, 00,00
BREAK IN 215
READY
```

Die falsche Aufzeichnungslänge in der DOPEN#-Anweisung in Zeile 210 hat diesen Fehler verursacht. BASIC 4.0 erlaubt nicht das Öffnen einer Relativen Datei unter Angabe der falschen Aufzeichnungslänge.

SCHREIBEN VON ZEICHENKETTEN IN RELATIVE DATEIEN

Beim Schreiben von Zeichenketten in Relative Dateien können Sie jede Zeichenkette mit einem Komma oder einem RETURN-Symbol abschließen. Wenn Sie jede Zeichenkette mit RETURN abschließen, besteht eine Aufzeichnung jeweils aus einer Zeichenkette. Sie können aber eine Aufzeichnung auch aus einer Reihe von Zeichenketten bilden, wenn Sie als Trennsymbol zwischen den Zeichenketten das Komma verwenden. Die letzte Zeichenkette in der Aufzeichnung muß mit einem RETURN-Symbol abgeschlossen werden.

Für unser erstes Programmbeispiel, mit dem wir das Schreiben von Zeichenketten in Relative Dateien zeigen, werden wir das sequentielle Adreßlisten-Programm SEQ.ADR.LISTE.B4 abwandeln. Das abgewandelte Programm bildet eine Relative Datei, in der jede Adresse jeweils eine Aufzeichnung zu je 5 Feldern bildet. Dieses neue Programm trägt den Namen REL.ADR.LISTE.B4 und ist nachfolgend aufgelistet; Programmänderungen wurden durch Schattierung sichtbar gemacht:

REL.ADR.LISTE. B4 (BASIC 4.0-Version)

```
10 REM  PROGRAMM ** REL.ADR.LISTE.B4 **
20 REM  ADRESSLISTEN-PROGRAMM ZUR DEMONSTRATION DES
21 REM  ARBEITENS MIT RELATIVEN DATEIEN AUF DISKETTEN
30 DATA " NAME    : "," BRANCHE: "," STRASSE: "
31 DATA " STADT   : "," LAND    : "
40 DOPEN#1, "REL.ADR.LISTE",L50
50 IF DS<>0 THEN PRINT DS$ : STOP
60 PRINT"□ BITTE ADRESSE EINGEBEN :■■"
70 FOR I=1 TO 5
80 READ F$
90 PRINT F$;:INPUT AD$(I)
100 NEXT I
110 RESTORE
120 PRINT "■SPEICHERN? J EINGEBEN ** AENDERN? N EINGEBEN"
130 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 130
135 PRINT Y$
140 IF Y$="N" THEN 60
150 REM  ADRESSE AUF RELATIVE DATEI SCHREIBEN
160 CM$=CHR$(44)
170 PRINT#1,AD$(1);CM$;AD$(2);CM$;AD$(3);CM$;AD$(4);CM$;AD$(5)
171 IF DS<> 0 THEN PRINT DS$:STOP
180
190 PRINT "WEITERE ADRESS-EINGABEN? J ODER N EINGEBEN"
200 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 200
205 PRINT Y$
210 IF Y$="J" THEN 60
220 DCLOSE#1
224 IF DS<>0 THEN PRINT DS$ : STOP
300 REM  ADRESSEN EINZELN WIEDERGEBEN
310 DOPEN#1,"REL.ADR.LISTE",L50
330 IF DS<>0 THEN PRINT DS$: STOP
340 REM  BILDSCHIRM LOESCHEN & ADRESSE DARSTELLEN
350 PRINT"□■■"
360 RESTORE
365 INPUT#1,AD$(1),AD$(2),AD$(3),AD$(4),AD$(5)
366 IF DS<>0 THEN PRINT DS$ : STOP
370 FOR I=1 TO 5
380 READ F$: PRINT F$;
390
400
410 PRINT AD$(I)
420 NEXT I
```

```
430 PRINT "WEITERE ADRESSE DARSTELLEN? J ODER N EINGEBEN"
440 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 440
450 IF Y$="J" THEN 350
460 DCLOSE#1
470 SCRATCH D0,"REL.ADR.LISTE"
480 STOP
```

Laden Sie das Programm SEQ.ADR.LISTE.B4 von der Diskette, um die Programmänderungen an den schattierten Zeilen auszuführen. Wenn Sie die Programmänderungen richtig ausgeführt haben, und das Programm wie sein Original SEQ.ADR. LISTE.B4 arbeitet, dann sollten Sie es auf einer Diskette speichern.

Lassen Sie uns die Programmänderungen in REL.ADR.LISTE.B4 untersuchen.

Die Anweisungen DOPEN# in Zeilen 40 und 310 wurden derart geändert, daß Sie jetzt eine Relative Datei mit Aufzeichnungen von 50 Zeichen Länge und dem Dateinamen REL.ADR.LISTE definieren.

Die Dateneingabe über das Tastenfeld und die Adreßdarstellung auf dem Bildschirm sind wie beim Programm SEQ.ADR.LISTE.B4. Das Schreiben der Adresse in die Relative Datei benutzt jedoch völlig andere Anweisungen. Die PRINT#-Anweisung in Zeile 170 schreibt eine einzelne Aufzeichnung. Das Komma wird in der Parameterliste der PRINT#-Anweisung mit der Variablen CM$ dargestellt, der in Zeile 160 das Komma-Symbol mit CHR$ (44) zugeordnet wurde. Beachten Sie auch die Semikolons als Trennsymbole zwischen jeder Variablen in der Parameterliste der PRINT#-Anweisung. Semikolon und Komma in Form von CM$ führen zu folgender Form der Adreßaufzeichnung in einer Relativen Datei:

```
170 PRINT#1,AD$(1);CM$;AD$(2);CM$;AD$(3);CM$;AD$(4);CM$;AD$(5)
```

Dieses Beispiel nimmt folgende Gleichsetzungen an: AD$ (1) = "Peter Stern" AD$ (2) = "Ingenieur", AD$ (3) = "Talstraße 1", AD$ (4) = "Schwaz" und AD$ (5) = "AUS".

Beachten Sie die Anweisungen in Zeile 171, mit denen die Diskette nach jedem Schreibvorgang auf ihren Status geprüft wird. Streng genommen hätte auch das Programm SEQ.ADR.LISTE.B4 derartige Anweisungen zum Prüfen des Disketten-Status enthalten sollen; das wäre gute Programmierpraxis gewesen. Unbedingt notwendig sind derartige Status-Prüfungen für ein Programm, das wie das vorliegende, Daten in Relative Dateien schreibt, um das Überschreiten der Aufzeichnungslänge durch zu große Datenmengen in Form einer Überlauf-Meldung zu erkennen. Ohne die Status-Prüfung in Zeile 171 würde jede Adresse, die die zulässige Aufzeichnungslänge überschreitet, falsch gespeichert werden; wenn Ihr Auge dazu fähig wäre, könnten Sie die Fehlerlampe am Floppy Disk-Speicher rot aufblitzen sehen, wenn eine zu große Datenmenge zum Überlauf in einer Aufzeichnung führt. Es gibt sonst keine Anzeichen dafür, daß ein Überlauf sich ereignet hat, bis Ihr Programm beim Lesen der Datei feststellt, daß eins oder mehrere Felder in einer Aufzeichnung fehlen.

Wir können die Notwendigkeit für die Status-Prüfung in Zeile 171 dadurch demonstrieren, daß wir diese Programmzeile löschen und die Semikolons in Zeile 170 in Kommas umändern. Bei erneutem Programmablauf können Sie vielleicht die Fehleranzeige am Floppy Disk-Speicher rot aufleuchten sehen, während Aufzeichnungen auf die Diskette geschrieben werden. Bei der nachfolgenden Adreßdarstellung auf dem Bildschirm werden die ersten zwei oder drei Felder jeder Adresse vorhanden sein, während die übrigen fehlen.

Was war der Grund?

Die Wirkung von Kommas in der Parameterliste der PRINT#-Anweisung in Zeile 170 auf die Felder einer Relativen Datei ist die gleiche wie auf eine Bildschirmdarstellung: Jedes Feld wird gegenüber dem vorigen Feld um 10 Zeichen versetzt geschrieben bzw. auf dem Bildschirm dargestellt. Die Parameterliste der PRINT#-Anweisung in Zeile 170 besteht aus 9 Variablen, die vier Variablen CM$ eingeschlossen. Da Kommas zwischen den Variablen 10 Zeichen Abstand zwischen jeder Variablen verursachen, muß eine Aufzeichnung wenigstens 90 Zeichen umfassen. Noch mehr Zeichen sind erforderlich, wenn die fünf Felder der Adresse aus mehr als 10 Zeichen bestehen. Die Wirkung der Kommas können Sie auf dem Bildschirm sichtbar machen, wenn Sie die folgende Anweisung in das Programm REL.ADR.LISTE.B4 einfügen:

```
170 PRINT#1,AD$(1),CM$,AD$(2),CM$,AD$(3),CM$,AD$(4),CM$,AD$(5)
```

Mit dieser Anweisung wird jede Aufzeichnung in der Form auf dem Bildschirm dargestellt, in der sie auch in die Relative Datei REL.ADR.LISTE. geschrie-

Zeilen 365 und 366 zeigen Anweisungen, mit denen die Adresse von der Relativen Datei zurückgelesen wird. Die entsprechenden Anweisungen aus Programm SEQ.ADR.LISTE.B4 in Zeilen 390 und 400 wurden entfernt.

Jede INPUT#-Anweisung liest eine Aufzeichnung von der Diskette zurück. Dies gilt für alle INPUT#-Anweisungen, gleichgültig von welchem Datei-Typ auf der Diskette gelesen wird. Mit anderen Worten: Jede INPUT#-Anweisung liest Daten von einem RETURN-Symbol bis zum nächsten RETURN-Symbol. Im Programm REL.ADR.LISTE.B4 liegen fünf Felder zwischen je zwei RETURN-Symbolen, so daß die INPUT#-Anweisung in Zeile 365 bei jedmaliger Ausführung die fünf Felder AD$ (1) bis AD$ (5) liest. Das Lesen der fünf Felder erfolgt auch dann, wenn die Parameterliste der INPUT#-Anweisung nicht aus genau fünf Variablen besteht.

Die INPUT#-Anweisung in Zeile 365 enthält fünf Zeichenketten-Variablen ihrer Parameterliste. Wenn eine der Variablen in der Parameterliste nicht eine Zeichenketten-Variable darstellte, würden Sie einen Syntax-Fehler erhalten und das Programm seinen Ablauf unterbrechen.

Wenn weniger als fünf Zeichenketten-Variable in der Parameterliste auftreten, dann werden einige der Variablen gegen Ende der Aufzeichnung zwar gelesen, aber nicht gespeichert. Sie können diesen Vorgang sichtbar machen, wenn Sie AD$ (4) und AD$ (5) aus der Parameterliste der INPUT#-Anweisung in Zeile 365 löschen. Bei erneutem Ablauf des Programms werden die ersten drei Felder der Adresse richtig wiedergegeben, die letzten zwei Felder fehlen jedoch.

Fügen Sie als nächsten Versuch die Variable AD$ (6) am Ende der Parameterliste INPUT#-Anweisung in Zeile 365 hinzu. Beim Programmlauf werden Sie feststellen, daß diese zusätzliche Variable ohne Auswirkung bleibt. Anders als bei Sequentiellen Dateien fehlen der zusätzlichen Variablen zugeordnete Daten, da die Felder der Aufzeichnung bereits von den anderen Variablen gelesen worden sind.

DIREKTER ZUGRIFF AUF EINZELNE AUFZEICHNUNGEN

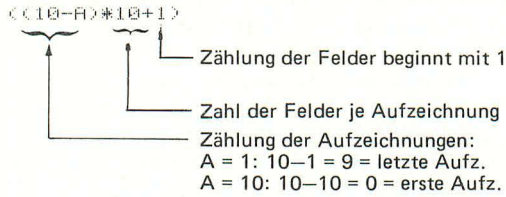
Die Anweisung RECORD# ermöglicht den direkten Zugriff auf jede Aufzeichnung und jedes Zeichen (Byte) innerhalb einer Aufzeichnung in einer Relativen Datei. Der Zugriff kann dazu dienen, eine Aufzeichnung oder ein Zeichen zu lesen oder mit einer neuen Information zu überschreiben.

Fügen Sie die Anweisung RECORD# auf Zeile 240 des Programms REL. NUM.B4 ein:

```
240 RECORD#1,((10-A)*10+1)
```

Jedes Feld der vom Programm REL.NUM.B4 angelegten Relativen Datei wird von RECORD# der Reihe nach für einen Zugriff markiert. Im vorliegenden Fall besteht der Zugriff aus dem Lesen jedes Feldes mit Anweisung INPUT# in Zeile 260. Im Gegensatz zum Programm bewirkt RECORD# mit seinem unten erläuterten Argument, daß die Felder der Relativen Datei in umgekehrter Reihenfolge dargestellt werden. Die 10 dargestellten Aufzeichnungen beginnen mit den Zahlen 901 bis 910 und enden mit den Zahlen 1 bis 10.

Das Argument in RECORD# gibt an, auf das wievielte Feld in der Relativen Datei zugegriffen werden soll, so daß sich sein Aufbau wie folgt erklärt:



Der Zugriff auf Aufzeichnungen und Felder in Relativen Dateien durch RECORD# ist unabhängig von deren Inhalt, d.h. ob es sich um numerische Daten oder Zeichenketten handelt. Im eben behandelten Anwendungsfall enthielt jede Aufzeichnung ein Feld. Bei Zeichenketten kann eine Aufzeichnung mehrere Felder enthalten; zur Bezeichnung jedes Feldes nimmt das Argument in RECORD# eine andere Form an, die ausführlich in Kapitel 8 beschrieben ist.

ÄNDERUNG DES INHALTS EINES FELDES

Die Anweisung RECORD# ermöglicht den Zugriff auf jede Aufzeichnung und jedes Zeichen einer Relativen Datei. Der Zugriff kann auch im Schreiben einer neuen Information in eine Aufzeichnung oder an eine Zeichenstelle bestehen. Besondere Programmiertechniken sind hierzu nicht erforderlich. Die gleiche PRINT#-Anweisung, mit der ein Feld geschrieben wurde, kann auch zum Überschreiben des Feldes verwendet werden, nachdem es im Argument einer vorangegangenen RECORD#-Anweisung bezeichnet worden ist.

GEBRAUCH VON GET# FÜR DISKETTEN-DATEIEN

Eine GET#-Anweisung liest jeweils ein Zeichen aus einer Daten-Datei auf Diskette, genauso wie die GET-Anweisung jeweils ein Zeichen vom Tastenfeld liest. Das Lesen eines Zeichens mit GET# erfolgt aus dem 256 Byte umfassenden Pufferspeicher der Diskette. Die Zeichen werden sequentiell gelesen, beginnend mit dem ersten Zeichen im Pufferspeicher. Leerstellen, Symbole der Zeichensetzung und alles, was eine Zeichenposition belegt, wird von GET# gelesen.

Sequentielle Dateien werden mit der Anweisung GET# sequentiell gelesen, beginnend mit dem ersten Zeichen in der Datei.

Relative Dateien können mit der Anweisung GET# an beliebigen Stellen gelesen werden, wenn eine vorangehende RECORD#-Anweisung den nachfolgenden Zugriff von GET# auf das Zeichen vorbereitet hat.

Vermeiden Sie den Gebrauch von GET# zum Lesen numerischer Daten von Disketten-Dateien. Zur Erinnerung: Eine GET#-Anweisung interpretiert das Nullzeichen als numerische Null. Die Unterscheidung zwischen numerischer Null und Nullzeichen geht verloren. (Nullzeichen: Siehe Anhang A, Tabelle A-4, ASCII-Kode 32).

Zur Veranschaulichung der Anwendung von GET# werden wir die Programme SEQ.ADR.LISTE.B4 und REL.RDR.LISTE.B4 abwandeln, indem wir Adressen statt mit einer INPUT#-Anweisung mit GET# zurücklesen. Die gleichen Änderungen sind auch gültig für das Programm SEQ.ADR.LISTE.B3.

GET# BEI SEQUENTIELLEN DATEIEN

Wir werden zuerst das Programm SEQ.ADR.LISTE.B4 abwandeln, indem wir die INPUT#-Anweisung in Zeile 390 durch GET# ersetzen. Die Anwendung von GET# folgt den gleichen Regeln, die für GET gelten. Hier die neue Programmzeile 390:

```
390 GET#1,AD$:IF AD$="" THEN 390
```

Die PRINT-Anweisung in Zeile 410 stellt jetzt jeweils nur ein Symbol dar; wir müssen daher die Variable in der PRINT-Anweisung mit einem Semikolon abschließen, um einen Zeilensprung (RETURN) zu unterdrücken.

Das Auftreten eines RETURN-Symbols in der gelesenen Datei muß mit einer zusätzlichen Anweisung auf einer neuen Programmzeile 415 erkannt werden:

```
415 IF AD$<>CHR$(13) THEN 390
```

Die IF-Anweisung in Zeile 415 verzweigt zurück zur GET#-Anweisung in Zeile 390, bis in der gelesenen Datei ein RETURN-Symbol entdeckt wurde. Erst dann kann ein weiterer Zyklus der FOR-NEXT-Schleife stattfinden. Das RETURN-Symbol am Ende jeder Aufzeichnung wird zuerst durch die PRINT-Anweisung in Zeile 410 auf dem Bildschirm dargestellt, ehe die IF-Anweisung in Zeile 415 die Programmlogik zum Lesen der nächsten Aufzeichnung veranlaßt.

Laden Sie das Programm SEQ.ADR.LISTE.B4 in den Speicher des Computers und fügen Sie die Änderungen ein. Das geänderte Programm sollte die gleichen Ergebnisse liefern.

Sie können weitere Versuche mit der GET#-Anweisung machen, die Sie dazu benutzen können, nach dem Auftreten eines bestimmten Zeichens in der gelesenen Datei zu suchen. Beispielsweise können Sie nach einem RETURN-Symbol suchen, das Sie mittels einer IF-THEN-Anweisung in ein graphisches Symbol umwandeln und auf dem Bildschirm darstellen.

GET# BEI RELATIVEN DATEIEN

Das unten aufgelistete Programm entstand aus einer Abwandlung des Programms REL.ADR.LISTE.B4 durch Übergang von INPUT# auf GET#.

REL.ADR.LISTE.GET #B4 (BASIC 4.0-Version)

```
10 REM  PROGRAMM ** ADR.LISTE.GET# **
20 REM  ADRESSLISTEN-PROGRAMM ZUR DEMONSTRATION DES
21 REM  ARBEITENS MIT RELATIVEN DATEIEN AUF DISKETTEN
30 DATA " NAME    : "," BRANCHE: "," STRASSE: "
31 DATA " STADT   : "," LAND    : "
40 DOPEN#1, "REL.ADR.LISTE",L50
50 IF DS<>0 THEN PRINT DS$ : STOP
60 PRINT"■ BITTE ADRESSE EINGEBEN :■■"
70 FOR I=1 TO 5
80 READ F$
90 PRINT F$;:INPUT AD$(I)
100 NEXT I
110 RESTORE
120 PRINT "■SPEICHERN? J EINGEBEN ** AENDERN? N EINGEBEN"
130 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 130
135 PRINT Y$
140 IF Y$="N" THEN 60
150 REM  ADRESSE AUF RELATIVE DATEI SCHREIBEN
160 CM$=CHR$(44)
170 PRINT#1,AD$(1);CM$;AD$(2);CM$;AD$(3);CM$;AD$(4);CM$;AD$(5)
171 IF DS<> 0 THEN PRINT DS$:STOP
190 PRINT "WEITERE ADRESS-EINGABEN? J ODER N EINGEBEN"
200 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 200
205 PRINT Y$
210 IF Y$="J" THEN 60
220 DCLOSE#1
224 IF DS<>0 THEN PRINT DS$ : STOP
300 REM  ADRESSEN EINZELN WIEDERGEBEN
310 DOPEN#1,"REL.ADR.LISTE",L50
330 IF DS<>0 THEN PRINT DS$: STOP
340 REM  BILDSCHIRM LOESCHEN & ADRESSE DARSTELLEN
350 PRINT"■■"
360 RESTORE
370 FOR I=1 TO 5
380 READ F$: PRINT F$;
390 GET#1,AD$ : IF AD$="" THEN 390
395 IF DS<>0 THEN PRINT DS$ : STOP
400 IF AD$=CHR$(32) THEN AD$="*"
405 IF AD$=CHR$(44) THEN PRINT AD$; : AD$=CHR$(13)
410 PRINT AD$;
415 IF AD$<>CHR$(13) THEN 390
420 NEXT I
430 PRINT "■WEITERE ADRESSE DARSTELLEN? J ODER N EINGEBEN"
440 GET Y$: IF Y$<>"J" AND Y$<>"N" THEN 440
450 IF Y$="J" THEN 350
460 DCLOSE#1
470 SCRATCH D0,"REL.ADR.LISTE"
480 STOP
```

Da eine GET#-Anweisung jeweils ein Zeichen liest, haben wir uns nicht um die verschiedenen Zeichensetzungen zu kümmern, mit denen Aufzeichnungen und Felder voneinander getrennt werden. Die GET#-Anweisung liest die Symbole der Zeichensetzung wie jedes andere Symbol, ohne sich durch die Bedeutung der Symbole im Lesen unterbrechen zu lassen. Daher konnten die INPUT#- und Status-Test-Anweisungen in Zeilen 365 und 366 des Programms REL.ADR.LISTE.B4 entfernt werden. In Zeile 390 wurde eine GET#-Anweisung eingefügt, zu deren Operation eine Status-Prüfung in der IF-Anweisung in Zeile 395 erfolgt.

Um Leerstellen auf dem Bildschirm sichtbar zu machen, werden in Zeile 400 Leerstellen durch das auffälligere (*)-Symbol dargestellt.

Das Auftreten eines Kommas wird in Zeile 405 erkannt. Jedes Komma wird zunächst dargestellt und dann durch einen Zeilensprung (RETURN) ersetzt.

Da die PRINT-Anweisung in Zeile 410 jetzt nur jeweils ein Symbol darstellt, mußte sie zur Vermeidung eines Zeilensprungs mit Semikolon abgeschlossen werden. In Zeile 415 verzweigt die Programmlogik zurück auf die GET#-Operation für das nächste Zeichen, bis ein RETURN-Symbol entdeckt wurde; dies ist das Zeichen für den Beginn einer neuen Aufzeichnung und eines neuen Zyklus in der FOR-NEXT-Schleife. Da in Zeile 405 Kommas in Zeilensprünge umgewandelt wurden, führen jetzt sowohl Kommas wie RETURN-Symbole in Zeile 415 zum Übergang auf das nächste Feld in der Adreßaufzeichnung.

Nach Durchführung der Programmänderungen werden Sie bei einem Programmlauf feststellen, daß sich die gleichen Bildschirmdarstellungen wie durch Programm REL.ADR.LISTE.B4 ergeben, abgesehen von Leerstellen, die jetzt als Sternsymbole dargestellt werden.

GET# UND RECORD# MIT RELATIVEN DATEIEN

Die RECORD#-Anweisung erlaubt, in jeder Aufzeichnung einer Relativen Datei jedes der gespeicherten Zeichen für einen nachfolgenden Zugriff zu markieren. Fügen Sie zur Veranschaulichung dieser Fähigkeit folgende Zeile in das letzte Programm ein:

```
365 RECORD#1,2,5
```

Der Zugriff auf die Datei durch GET# in Zeile 390 beginnt jetzt in Aufzeichnung 2 mit dem fünften Zeichen. (Hierzu muß die Relative Datei zumindest zwei Aufzeichnungen enthalten). Auf dem Bildschirm wird die zweite Adresse dargestellt, beginnend mit dem fünften Zeichen.

PROGRAMM-DATEIEN

CBM-Computer behandeln Dateien für Programme und Daten völlig verschieden. Jeder Dateityp verfügt über eigene Anweisungen zum Arbeiten mit diesen Dateien.

LADEN UND SPEICHERN VON PROGRAMM-DATEIEN

Programm-Dateien werden mit LOAD (für BASIC<3.0) oder mit DLOAD (für BASIC 4.0) von der Diskette in den Computer geladen. Programm-Dateien werden mit SAVE (für BASIC<3.0) oder mit DSAVE (für BASIC 4.0) vom Speicher des Computers auf die Diskette geschrieben.

Laden und Speichern von Programmen wurde ausführlich in Kapitel 2 beschrieben.

ARBEITEN MIT PROGRAMM-DATEIEN WIE MIT DATEN-DATEIEN

Sie können beim Arbeiten mit Dateien für Programme die gleichen Anweisungen OPEN, CLOSE, GET#, INPUT# und PRINT# benutzen, die Sie vom Arbeiten mit Dateien für Daten gewohnt sind. Ehe Ihnen aber das Programmsystem in CBM-Computern nicht ganz vertraut ist, können Sie zu unerwarteten Ergebnissen kommen. Außerdem würden Sie nicht mehr erreichen, als mit den Standard-Anweisungen für das Arbeiten mit Programm-Dateien und für die Bildschirm-Edition.

Beim Arbeiten mit BASIC<3.0 wird das Laden eines Programms (LOAD) mit der Sekundäradresse Null gekennzeichnet, während zum Speichern eines Programms (SAVE) die Sekundäradresse 1 verwendet wird. Verwendet man diese Sekundäradressen auch in OPEN-Anweisungen, dann spielt sich der Zugriff auf Programm-Dateien wie ein Zugriff auf Daten-Dateien ab.

RESERVE-PROGRAMM-DATEIEN

Es ist unbedingt erforderlich, sich von einem wichtigen Programm wenigstens eine oder zwei Kopien anzufertigen. Wann immer es möglich ist, sollten Sie wenigstens eine Kopie eines für Sie wichtigen Programms auf einer zweiten Diskette in Reserve haben

In BASIC 4.0 verwenden Sie zum Kopieren eines Programms die Anweisung COPY. Zum Kopieren einer ganzen Diskette steht die Anweisung BACK-UP zur Verfügung.

In BASIC<3.0 müssen Sie zum Kopieren von Programmen und Disketten eine Variation der Anweisung PRINT# verwenden, wie am Anfang dieses Kapitels beschrieben wurde.

ORGANISATION VON PROGRAMMENTWICKLUNGEN

Programme verändern sich unentwegt durch Korrekturen oder Verbesserungen. Sicherstes Vorgehen für Programmänderungen ist, eine Kopie der augenblicklichen Programmversion zusammen mit den beiden vorangegangenen Versionen aufzubewahren. Die Vorgänger der augenblicklichen Programmversion werden häufig "Vater" und "Großvater" genannt. Führen Sie die Änderung eines Programms in folgenden Stufen durch:

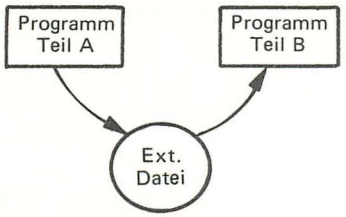
1. Laden (LOAD) der augenblicklich benutzten Programmversion in den Computer, um die beabsichtigten Änderungen vorzunehmen.
2. Löschen (SCRATCH) des augenblicklichen "Großvater"-Programms.
3. Neubenennung (RENAME) des "Vater"-Programms als "Großvater"
4. Neubenennung (RENAME) des augenblicklich benutzten Programms in "Vater".
5. Speichern (SAVE) der neuen Programmversion.

PROGRAMM-WARTESCHLANGEN

Wenn Sie Sorgfalt in der Vergabe von Zeilennummern zeigen, dann können Sie innerhalb eines Programms die Anweisungen LOAD und SAVE verwenden, um

sehr lange Programme zum Ablauf zu bringen und verschiedene Formen von Programm-Warteschlangen anzuwenden.

Stellen Sie sich den Fall vor, daß Ihr Programm zu lang geworden ist, um in den Speicherplatz Ihres Computers zu passen. Versuchen Sie, dieses Problem durch Aufteilen des Programms in zwei Teile zu lösen. Beide Programmteile müssen völlig unabhängig voneinander ablauffähig sein; ausgenommen sind Daten, die von dem einen Programmteil über eine externe Datei zum anderen Programmteil gelangen können. Das kann man sich wie folgt veranschaulichen:



Keine Anweisung in dem einen Programmteil kann auf Anweisungen in dem anderen Programmteil verzweigen.

Lassen Sie uns die beiden Programmteile Teil A und Teil B nennen. Ein kurzes Überwachungsprogramm kann in folgenden Schritten erst den Programmlauf von Teil A, dann von Teil B veranlassen:

1. Teil A in den Computer laden (LOAD).
2. Teil A ablaufen lassen.
3. Teil B in den Computer laden (LOAD).
4. Teil B ablaufen lassen.

Hier die Anweisungen eines kurzen Überwachungsprogramms in BASIC 4.0:

```
60000 DLOAD D0,"PROGRAMMTEIL A"
60010 REM  PROGRAMMTEIL A BEGINNT IN ZEILE 20
60020 GOSUB 20
60030 DLOAD D0,"PROGRAMMTEIL B"
60040 REM  PROGRAMMTEIL B BEGINNT IN ZEILE 50
60050 GOSUB 50
60060 END
```

Teil A muß seinen Lauf mit der Ausgabe von Daten in eine externe Datei beenden, aus der Teil B alle erforderlichen Daten entnehmen kann. Teil B muß seinen Ablauf damit beginnen, die von Teil A hinterlassenen Daten aus einer externen Datei in den Computer zu laden.

Wenn Sie diese Technik der Programmteilung anwenden, müssen Sie darauf achten, daß die Zeilennummern in Teil B beim Laden des Programms in den Computer diejenigen Zeilennummern ersetzen, die Teil A zuvor im Computer benutzt hat. Andererseits muß sicher sein, daß die Zeilennummern des Überwachungsprogramms erhalten bleiben. Um diesen Hinweis zu verstehen, erinnern wir uns, daß beim Laden eines neuen Programms in den Speicher des Computers der frühere Speicherinhalt nicht gelöscht wird, wenn nicht eine NEW-Anweisung ausgeführt wurde; und genau dies ist nicht möglich, da sonst auch das Überwachungsprogramm gelöscht werden würde. Beim Einlesen neuer Programmanweisungen in den Speicher des Computers werden die Anweisungen des alten Programms überschrieben, sofern sie gleiche Zeilennummern haben; alle anderen Anweisungen des alten Programms bleiben unverändert im Speicher.

Das Überwachungsprogramm benutzt Zeilennummern, die weder von Teil A noch Teil B verwendet werden dürfen, da sie nach dem Überwachungsprogramm in den Computer geladen werden und hierbei das Überwachungsprogramm löschen würden.

PROGRAMMIERUNG VON CBM-DRUCKERN

Bis zu diesem Zeitpunkt haben wir sehr wenig von CBM-Druckern Gebrauch gemacht. Wir haben uns Programme auflisten lassen; aber hierfür war kein Programmieraufwand erforderlich. Die meisten Programme liefern ihre Ergebnisse in Form von Ausdrucken. Das Aussehen derartiger Ergebnisdarstellungen ist sehr bedeutend; die Ergebnisse werden benutzt, wenn sie leicht zu lesen sind. Eine schlecht gestaltete Darstellung wird beiseite gelegt. Glücklicherweise ist es leicht, gut gestaltete Ergebnisdarstellungen auf CBM-Druckern zu programmieren.

Für CBM-Computer-Systeme stehen folgende Drucker zur Verfügung: CBM 4022, CBM 8024 und die Modelle CBM 3022 und CBM 3023. Die Druckerfunktionen dieser Geräte werden durch eingebaute Mikroprozessoren gesteuert, womit sich erklärt, warum gut gestaltete Druckbilder mit diesen Geräten leicht zu programmieren sind.

Drucker werden über eine OPEN-Anweisung angesprochen, in der neben einer logischen Dateinummer die Geräteadresse für Drucker und eine Sekundäradresse zur Steuerung der Betriebsart des Druckers angegeben sind. Für Drucker gilt die Geräteadresse # 4. Als Sekundäradresse kann ein Zahlenwert zwischen Null und Sechs angegeben werden, womit folgende Betriebsarten des Druckers steuerbar sind:

0 Daten drucken wie empfangen.
1 Daten in einem zuvor mitgeteilten Format ausdrucken.
2 Informationen über das Druckformat empfangen.
3 Informationen über die Zahl der Zeilen/Seite empfangen.
4 Fehlermeldungen über falsche Formatangaben senden.
5 Informationen über ein Benutzer-entworfenes Druckzeichen empfangen.
6 Informationen über den Zeilenabstand empfangen.

Fehlt die Angabe einer Sekundäradresse, dann nimmt der Drucker als Ersatzwert 0. Die Betriebsart unter Sekundäradresse 0 ermöglicht, zahlreiche Druckerfunktionen durch Übermitteln von Steuerkodes zu wählen (siehe Tab. 6-4). Unter Steuerkode 2 können dem Drucker Informationen über die Druckbild-Gestaltung mitgeteilt werden, die zusammen mit Beispielen in Tabelle 6-5 beschrieben sind.

DATEN AUSDRUCKEN WIE EMPFANGEN

Zum Ausdrucken von Daten in der empfangenen Form verwenden Sie eine OPEN-Anweisung mit der Geräteadresse # 4. Eine Sekundäradresse ist nicht erforderlich, oder es wird als Sekundäradresse Null angegeben. Zur Übergabe der Daten an den Drucker verwenden Sie die Anweisungen PRINT# und/oder CMD.

DRUCKEN MIT DER PRINT#-ANWEISUNG

Die PRINT#-Anweisung übermittelt Daten an den Drucker genauso, wie sie Daten in eine Kassetten- oder Disketten-Datei schreibt. Ein Beispiel: Um das Wort "MESSAGE" auszudrucken, geben Sie folgendes Programm ein und lassen es ablaufen:

```
10 OPEN 2,4
20 PRINT#2,"MESSAGE"
30 CLOSE 2
40 STOP
```

Dies Programm veranlaßt bei jedem Lauf den Ausdruck des Wortes "MESSAGE"; danach erscheint folgende Nachricht auf dem Bildschirm:

```
BREAK IN 40
READY
```

Diese Nachricht wird von der STOP-Anweisung in Zeile 40 ausgelöst.

Die Anweisungen DOPEN# und DCLOSE# in BASIC 4.0 können nicht zum Arbeiten mit dem Drucker benutzt werden. Diese Anweisungen sind nur zum Arbeiten mit Disketten geeignet.

DRUCKEN MIT DER CMD-ANWEISUNG

Für die Übertragung von Daten zum Drucker können Sie statt der PRINT#-Anweisung auch die CMD-Anweisung verwenden. Der CMD-Anweisung muß jedoch wenigstens eine PRINT#-Anweisung folgen, ehe die logische Datei für den Drucker geschlossen werden kann. Folgendes Programm zeigt die Anwendung der CMD-Anweisung:

```
10 OPEN 2,4
20 CMD 2,"MESSAGE"
25 PRINT#2
30 CLOSE 2
40 STOP
```

Wenn Sie dieses Programm ablaufen lassen, wird das Wort "MESSAGE" gedruckt, gefolgt von zwei Wagenrückläufen. Der zweite Wagenrücklauf wird durch die PRINT#-Anweisung in Zeile 25 verursacht.

AUSDRUCKEN MIT DEN ANWEISUNGEN CMD UND PRINT

Nach einer CMD-Anweisung ändern PRINT-Anweisungen die Richtung ihrer Datenausgabe: Daten werden nicht mehr auf dem Bildschirm dargestellt, sondern an den Drucker übermittelt, bis im Programm eine PRINT#-Anweisung auftritt. Ändern Sie für einen Versuch das eben besprochene Drucker-Programm wie folgt:

```
10 OPEN 2,4
20 CMD 2
22 PRINT "MESSAGE"
24 PRINT#2
26 PRINT "MESSAGE"
30 CLOSE 2
40 STOP
```

Beim Ablauf dieses Programms führt der Drucker zunächst einen Wagenrücklauf aus, druckt dann das Wort "MESSAGE", gefolgt von zwei weiteren Wagenrückläufen und danach wird das Wort "MESSAGE" auf dem Bildschirm dargestellt. Die CMD-Anweisung in Zeile 20 löst den ersten Wagenrücklauf aus; die PRINT-Anweisung in Zeile 22 löst den Ausdruck des Wortes "MESSAGE" sowie den zweiten Wagenrück-

Tabelle 6-4. Steuerung von Drucker-Funktionen

+)	Drucker-Funktion	Steuerkode *)	Kode-gleiche Tasteneingabe	Erläuterungen – Anwendungen
A	Sprung zum nächsten formatierten Feld	CHR$ (29)	Kursor →	Format: `10 PRINT#1,"AA  AA  AA"` Daten: `20 PRINT#2,"XX"CHR$(29)"YY"CHR$(29)"ZZ"` Ausdruck: `XX  YY  ZZ`
	Leerstelle in formatierten Feldern	CHR$ (160)	SHIFT + Leertaste	Format: `10 PRINT#1,"AAAAAAA"  AAAAAAA` Text: `20 PRINT#2,"NAME"  NAME` Text m. führ. Leerstellen: `30 PRINT#2,"  NAME"  NAME`
B	Breitschrift	CHR$ (1)		`10 PRINT#1,"E" CHR$(1)"E" CHR$(1)"E"` `E  E  E`
	Zeilenvorschub	CHR$ (10)		
	Seitenvorschub	CHR$ (12)		Vorschub auf den Anfang der folgenden Seite
	Wagenrücklauf	CHR$ (13)	RETURN	Wagenrücklauf und Zeilenvorschub
	Negativschrift EIN	CHR$ (18)	OFF RVS	gelöscht vom nächsten Wagenrücklauf
	Kleinschreibung	CHR$ (17)	KURSOR ↓	gelöscht vom nächsten Wagenrücklauf oder Großschreibung-Kode CHR$ (145)
	Paging AUS	CHR$ (19)	HOME	Übergang auf nächsten Seitenbeginn
	Anführungszeichen	CHR$ (34)		Ausdruck aller Steuerkodes mit graphischen Symbolen
	Normalschrift	CHR$ (129)		Löschen des Breitschrift-Kodes CHR$ (1)
	Wagenrücklauf ohne Zeilenvorschub	CHR$ (141)		Wagenrücklauf ohne Zeilenvorschub: etwa vorhandener Text wird überschrieben
	Großschreibung	CHR$ (145)	KURSOR ↑	Ausdruck wechselt zu Großschreibung, wenn zuvor Kleinschreibung – CHR$ (17) – wirksam war
	Negativschrift AUS	CHR$ (146)	SHIFT + OFF RVS	Löschen des Negativschrift-Kodes CHR$ (18)
	Paging EIN	CHR$ (147)	CLR	Ausdruck erfolgt mit 66 Zeilen/Seite, bis Anweisung mit Sekundäradresse 3 eine andere Zeilenzahl/Seite vorschreibt

*) s. a. Anhang A, Tabelle A-4

+) A = nur bei formatierten Ausdrucken anwendbar. B = bei formatierten und unformatierten Ausdrucken anwendbar.

Tabelle 6-5. Formatierung von Ausdrucken

Daten Typ	Kode	Definition	Anwendung	Daten	Beispiele Format	Druckbild
	9	Zahl ohne führende Nullen	Jede Stelle vor und nach dem Dezimalpunkt wird mit einer "9" gekennzeichnet	23 124.756 124.756	999.9 9999.99 999	23.0 124,75 124
Numerische Felder	Z	Zahl mit führenden Nullen	Jede Stelle vor dem Dezimalpunkt wird mit "Z" statt "9" gekennzeichnet	23 124.756 124.756	ZZZ.9 ZZZZ.99 ZZZ	023.0 0124.75 124
	.	Dezimalpunkt	Zahlendarstellung mit übereinander liegenden Dezimalpunkten		siehe unten	
	$	Dollarzeichen	Ausdruck eines $-Zeichens vor der Zahlendarstellung	23 124.756	$999.9 $$$$$.99	$23.0 $124.75
	S	Vorzeichen +, –	Vorzeichen (+ oder –) unmittelbar vor einer Zahl	124.756 –23 –475.2	S$999.99 S$$$$.99 S9999.9	+$124.75 –$23.00 +475.2
	–	Minuszeichen hinter einer Zahl	Minuszeichen (–) unmittelbar hinter einer negativen Zahl	124.756 –23 –475.2	$999.99– $$$$.99– 9999.9–	$124.75 $23.00– 475.2–
Zeichenfelder	A	Buchstabe	Jede Stelle einer Buchstaben-Darstellung wird mit "A" gekennzeichnet	ABCDE ABC ABCDE	AAAAAA AAAAAA AA	ABCDE ABC AB
beliebig	∅	Nullzeichen	Zur Trennung von Druckfeldern. Dargestellt durch CHR$ (32).			
	<RVS ON>	Zeichendarstellung wie vorgegeben	Das in der Formatangabe enthaltene Zeichen wird gedruckt		<RVS ON>– <RVS ON>– A	– –

lauf aus. Die PRINT#-Anweisung in Zeile 24 führt zum dritten Wagenrücklauf. Schließlich stellt die PRINT-Anweisung in Zeile 26 das Wort "MESSAGE" auf dem Bildschirm dar.

Wenn Sie jetzt die PRINT-Anweisung in Zeile 22 löschen, wird der Drucker zwei Wagenrückläufe ausführen, das Wort "MESSAGE" aber wird nicht ausgedruckt, sondern auf dem Bildschirm dargestellt.

EIN VERGLEICH DER ANWEISUNGEN CMD UND PRINT#

Um die unterschiedliche Wirkung der PRINT-Anweisung im letzten Programmbeispiel zu erklären, müssen wir die Funktion einer CMD-Anweisung erläutern.

Stellen Sie sich vor, der Drucker ersetzt den Bildschirm. Der Ausgabekanal des CBM-Computers geht entweder zum Bildschirm oder zum Drucker. Mit einer OPEN-Anweisung und der Geräteadresse #4 wird dem CBM-Computer mitgeteilt, daß ein Drucker vorhanden ist, der Ausgabekanal bleibt aber vorerst mit dem Bildschirm verbunden.

Folgt jetzt eine PRINT#-Anweisung, dann wird der Ausgabekanal vom Bildschirm auf den Drucker umgeschaltet. Die Daten aus der Parameterliste der PRINT#-Anweisung werden zum Drucker gesandt und danach der Ausgabekanal wieder zum Bildschirm zurückgeschaltet.

Nach einer CMD-Anweisung wird zwar auch der Ausgabekanal vom Bildschirm auf den Drucker umgeschaltet und die Daten in der Parameterliste der CMD-Anweisung zum Drucker übertragen; der Ausgabekanal bleibt danach aber mit dem Drucker verbunden und der Bildschirm ist nicht mehr an den Ausgabekanal angeschlossen.

Folgt jetzt eine PRINT-Anweisung, dann werden die Daten ausgedruckt und nicht auf dem Bildschirm dargestellt, da die CMD-Anweisung den Ausgabekanal vom Bildschirm auf den Drucker umgeschaltet hat. Folgt jedoch eine PRINT#-Anweisung, dann wird der Ausgabekanal am Ende der Befehlsausführung wieder mit dem Bildschirm verbunden. Eine PRINT-Anweisung nach einer PRINT#-Anweisung führt wieder zu Darstellungen auf dem Bildschirm.

Wie bei jeder anderen logischen Datei muß auch beim Drucker der Ausgabekanal wieder geschlossen werden. Eine CLOSE-Anweisung teilt dem CBM-Computer mit, daß der Datenverkehr zum Drucker aufgehoben werden soll. Nach einer CMD-Anweisung ist wenigstens eine PRINT#-Anweisung erforderlich, um die Schließung des Ausgabekanals zum Drucker durch CLOSE wirksam werden zu lassen.

Bleibt der Ausgabekanal mit dem Drucker verbunden, da zwischen CMD und CLOSE eine PRINT#-Anweisung fehlte, dann werden Daten nachfolgender PRINT-Anweisungen weiterhin vom Drucker dargestellt. Machen Sie hiermit Erfahrung, indem Sie folgendes Programm eingeben:

```
10 OPEN 2,4
20 CMD 2
30 CLOSE 2
35 PRINT "MESSAGE"
40 STOP
```

Der Ablauf dieses Programms führt zu folgendem Ausdruck:

```
MESSAGE
BREAK IN 40
READY
```

Trotz der CLOSE-Anweisung in Zeile 30 blieb der Ausgabekanal mit dem Drucker verbunden, so daß die üblicherweise auf dem Bildschirm erscheinenden Nachrichten BREAK und READY zum Drucker gesandt wurden.

FORMATIERTE AUSDRUCKE

Ein System aus CBM-Computer und CBM-Drucker kann Ausdrucke automatisch formatiert darstellen. Sie haben nur das Druckerformat festzulegen, und hierzu benutzen Sie die Regeln in Tabelle 6-5. Die Format-Information wird über eine OPEN-Anweisung mit der Sekundäradresse 2 und PRINT#-Anweisungen dem Drucker mitgeteilt.

Der Datenausdruck im festgelegten Druckerformat erfolgt dann über eine OPEN-Anweisung mit der Sekundäradresse 1 und wie gewohnt, durch PRINT#-Anweisungen. Bei fehlender Formatangabe nach einer Sekundäradresse 1 werden Daten in der übermittelten Form ausgedruckt – wie bei Ausdrucken mit der Sekundäradresse 0.

Die Programmierung formatierter Ausdrucke erfordert demnach zwei OPEN-Anweisungen: Die eine OPEN-Anweisung öffnet einen Kanal zur Übertragung der Format-Angaben (Geräteadresse 4, Sekundäradresse 2), die andere OPEN-Anweisung öffnet einen Kanal zur Übertragung auszudruckender Daten mit dem Hinweis an den Drucker, beim Ausdruck die Formatangaben zu beachten (Geräteadresse 4, Sekundäradresse 1). Die eigentliche Übertragung der Informationen erfolgt jeweils über PRINT#-Anweisungen mit den entsprechenden logischen Dateinummern.

FORMATIERTER AUSDRUCK NUMERISCHER DATEN

Zu Anfang werden wir untersuchen, wie der Drucker formatierte numerische Daten ausgeben kann. Die Stellung jedes Zeichens in einem Zahlenausdruck wird in einer Formatanweisung mit den Symbolen 9, Z und "." festgelegt.

Der Dezimalpunkt "." wird dort gedruckt, wo er in der Formatanweisung steht. In tabellenartigen Ausdrucken werden Zahlen automatisch mit übereinanderliegenden Dezimalpunkten dargestellt.

Die Ziffer 9 und der Buchstabe Z dienen beide zur Angabe der Stellung einer Zahl. Die Ziffer 9 stellt führende Nullen durch Leerstellen dar, wogegen der Buchstabe Z führende Nullen ausdruckt. Hierfür einige Beispiele:

Eingabe	Format-Angabe	resultierendes Druckbild
123.456 6457 −128.1	999999.99	123.45 6457.00 128.10
123.456 6457 −128.1	ZZZZZ.9	00123.4 00123.4 00128.1

Eine Zahl kann mit führendem oder nachfolgendem Vorzeichen ausgedruckt werden. Ein führendes Vorzeichen + oder − wird ausgedruckt, wenn am Anfang der Formatanweisung der Buchstabe S steht.

Ein nachfolgendes Minuszeichen (−) wird ausgedruckt, wenn die Formatanweisung mit einem Minuszeichen abgeschlossen ist; es gibt keinen Ausdruck nachfolgender Pluszeichen.

Ein Dollarzeichen ($) wird ausgedruckt, wenn in der Formatanweisung ein $ direkt vor dem Zahlenformat steht.

Hier sind einige Beispiele für Zahlenformate, mit denen ein Vorzeichen und/oder $ ausgedruckt werden:

Eingabe	Format-Angabe	resultierendes Druckbild
123.456 6457 −128.1	S9999	123 6457 −128
123.456 6457 −128.1	S$9999.99	$0123.45 $6457.00 −$0128.10
123.456 6457 −128.1	S$$$$$.99	$123.45 $6457.00 −128.10
123.456 6457 −128.1	$$$$$.99−	$123.45 $6457.00 $ 128.10−
123.456 6457 −128.1	$ZZZZ,99−	$0123.45 $6457.00 $0128.10−

Das Dollarzeichen ($) kann durch beliebige andere Symbole ersetzt werden, deren Programmierung wir gegen Ende dieses Kapitels beschreiben.

Für Versuche mit formatierten numerischen Ausdrucken können Sie das unten aufgelistete Programm NUM.FORM.PRINT eingeben. In diesem Programm werden 8 verschiedene Zahlen aus der DATA-Anweisung in Zeile 30 in dem von PRINT# angegebenen Format in Zeile 100 ausgedruckt. Nachfolgend das Programm und das Ergebnis eines Programmlaufs:

NUM.FORM.PRINT

```
10 REM PROGRAMM ** ZAHLEN-AUSDRUCK **
20 REM ZUR DEMONSTRATION FORMATIERTER ZAHLEN-AUSDRUCKE
30 DATA 1.75,-12300,0.74682,12,-456.832,23456.78,-100.798,4789326
70 OPEN 1,4,1 : REM  ZAHLEN-AUSGABE UEBER KANAL 1
80 OPEN 2,4,2 : REM  FORMAT-AUSGABE UEBER KANAL 2
90 REM   AUSGABE DES FORMATS:
100 PRINT#2,"999999.99"
105 REM  AUSGABE DER DATEN:
110 FOR I=1 TO 8
120 READ N
```

```
130 PRINT#1,N
140 NEXT I
150 CLOSE 1
160 CLOSE 2
170 STOP
```

```
     1.75
 12300.00
      .74
    12.00
   456.83
 23456.78
   100.79
******.**
```

Bemerkenswert am Ausdruck ist folgendes: Alle Zahlen wurden mit übereinanderstehenden Dezimalpunkten ausgedruckt; statt der achten Zahl wurden Sternsymbole ausgedruckt als Fehlermeldung dafür, daß die in der Formatanweisung vorgesehene Stellenzahl überschritten wurde.

Ändern Sie jetzt die Formatanweisung in Zeile 100 durch Austausch der Ziffern 9 vor dem Dezimalpunkt mit den Buchstaben Z. Ein Programmlauf liefert folgenden Ausdruck:

```
000001.75
012300.00
000000.74
000012.00
000456.83
023456.78
000100.79
******.**
```

Der Buchstabe Z brachte den Ausdruck führender Nullen. Die achte Zahl verursachte wie zuvor einen Überlauf. Wenn Sie einen weiteren Buchstaben Z vor dem Dezimalpunkt in der Formatanweisung einfügen, wird auch die achte Zahl ausgedruckt. Machen Sie einen Versuch.

Vor dem Dezimalpunkt in einer Formatanweisung dürfen Buchstaben Z und Ziffern 9 nicht gemischt auftreten. Andernfalls anerkennt der Drucker nur Symbole gleichen Typs am Anfang der Formatanweisung. Ein Beispiel: Ändern Sie die PRINT#-Anweisung in Zeile 100 wie folgt:

```
100 PRINT#2,"ZZZ2999.99"
```

Ein Programmlauf zeigt jetzt Ausdrucke, als ob die Formatanweisung "ZZZZ" gelautet hätte. Ändern Sie dann die PRINT#-Anweisung in Zeile 100 wie folgt:

```
100 PRINT#2,"9999ZZZ.99"
```

Nach dieser Änderung ergibt sich ein Programmausdruck, als ob die Formatanweisung "9999" gelautet hätte.

Zahlen wurden bis jetzt ohne Vorzeichen ausgedruckt. Ändern Sie zum Ausdruck eines führenden Vorzeichens die PRINT#-Anweisung in Zeile 100 wie folgt:

```
100 PRINT#2,"$9999999.99"
```

Die Zahlen werden jetzt mit führendem Vorzeichen und ohne führende Nullen dargestellt:

```
+       1.75
-   12300.00
+        .74
+      12.00
-     456.83
+   23456.78
-     100.79
+4789326.00
```

Für die Darstellung nachfolgender Vorzeichen muß die PRINT#-Anweisung in Zeile 100 wie folgt geändert werden:

```
100 PRINT#2,"9999999.99-"
```

Dieser Programmlauf stellt Zahlen mit nachfolgendem Vorzeichen dar; ausgedruckt wird jedoch nur das Minuszeichen.

Beachten Sie: Von allen Zahlen werden nur soviele Stellen ausgedruckt, wie in der Formatangabe vorgesehen; der Drucker führt keine Auf- oder Abrundungen aus.

Wir werden jetzt Zahlen in Dollarbeträge umwandeln, indem wir ein $ vor die Formatanweisungen setzen. Wollen wir außerdem ein führendes Vorzeichen ausdrucken, dann muß die PRINT#-Anweisung in Zeile 100 wie folgt geändert werden:

```
100 PRINT#2,"$+9999999.99"
```

Ein Programmlauf liefert folgenden Ausdruck:

```
+$       1.75
-$   12300.00
+$        .74
+$      12.00
-$     456.83
+$   23456.78
-$     100.79
+$4789326.00
```

Beachten Sie, daß der Buchstabe S **vor** dem Dollarzeichen $ stehen muß. Andernfalls werden die Zahlen unformatiert ausgedruckt.

In Finanzberichten ist es üblich, negative Dollarbeträge durch ein nachfolgendes Minuszeichen zu kennzeichnen. Derartige Ausdrucke ergeben sich, wenn Sie den Buchstaben S durch ein Minuszeichen am Ende der Formatangabe ersetzen:

```
100 PRINT#2,"$9999999.99-"
```

Das Druckbild nimmt jetzt folgende Gestalt an:

```
$       1.75
$   12300.00-
$        .74
$      12.00
$     456.83-
$   23456.78
$     100.79-
$4789326.00
```

In üblichen Ausdrucken von Dollarbeträgen steht das Dollarzeichen unmittelbar vor der ersten numerischen Stelle; die entsprechende Formatanweisung enthält nur $-Zeichen vor dem Dezimalpunkt:

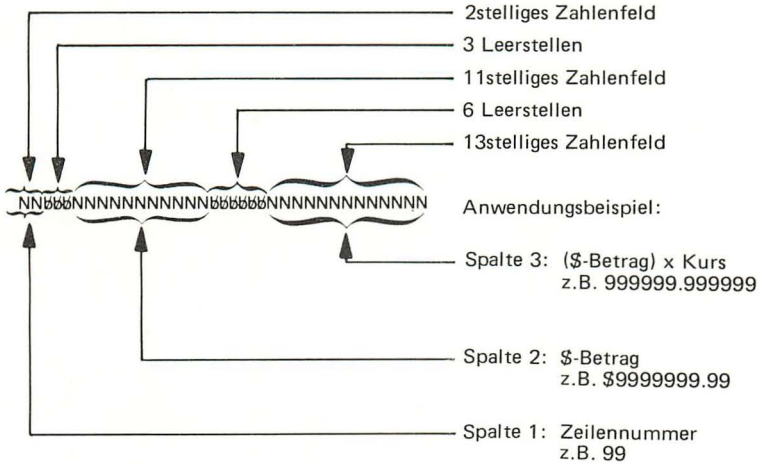
```
100 PRINT#2,"$$$$$$$.99-"
```

Dollarbeträge werden jetzt in folgender Gestalt ausgedruckt:

```
     $1.75
$12300.00-
     $.74
    $12.00
  $456.83-
$23456.78
  $100.79-
******.****
```

Die achte Zahl wurde wieder durch Sternsymbole ausgedruckt. Das hängt mit der ungenügenden Stellenzahl in der Formatanweisung zusammen: Für jede Ziffer vor dem Dezimalpunkt ist ein $ erforderlich, und zusätzlich ein $ für den eigentlichen Ausdruck des Dollarzeichens $.

Bis jetzt haben wir ein-spaltige Zahlenausdrucke behandelt. Zum mehr-spaltigen Ausdruck von Zahlen müssen die Formatanweisungen für jede Spalte derart durch Leerstellen getrennt werden, daß das gewünschte Tabellenformat entsteht. Ein Beispiel: Eine drei-spaltige Zahlendarstellung besteht wie folgt aus Formatangaben und Leerstellen:



Für diese drei-spaltige Zahlendarstellung muß die PRINT#-Anweisung in Zeile 100 folgende Form annehmen:

```
100 PRINT#2,"99   $$$$$$$$.99-    999999.999999"
```

Um unser Anwendungsbeispiel zu realisieren, ändern wir die PRINT#-Anweisung in Zeile 130 wie folgt:

```
130 PRINT#1,I,N,N/3
```

Ein Programmlauf erzeugt jetzt den folgenden Ausdruck:

```
1          $1.75              .583333
2      $12300.00-         4100.000000
3          $.74               .248940
4         $12.00              4.000000
5        $456.83-           152.277333
6      $23456.78          7818.926670
7        $100.79-            33.599333
8   $4789326.00        ******.******
```

Jede Zahlenspalte dieses Ausdrucks wurde entsprechend der Formatanweisung in Zeile 100 ausgedruckt. Der Abstand der Spalten zueinander hängt von der

Zahl der Leerstellen zwischen den Formatanweisungen des PRINT#-Befehls in Zeile 100 ab.

FORMATIERTER AUSDRUCK VON ZEICHENKETTEN

Zum formatierten Ausdruck von Zeichenketten werden Formatanweisungen verwendet, in denen mit dem Buchstaben A die Stellung jedes Zeichens im Ausdruck notiert wird und Leerstellen den Abstand zwischen Zeichenketten angeben. Diese Formatangabe nimmt die Form einer Parameterliste in einer PRINT#-Anweisung an. Wie früher beschrieben, muß diese PRINT#-Anweisung sich durch die gleiche logische Dateinummer auf diejenige OPEN-Anweisung beziehen, in der die Sekundäradresse 2 angegeben worden war.

Die formatiert auszudruckenden Zeichenketten werden mit einer weiteren PRINT#-Anweisung an den Drucker übertragen. Diese PRINT#-Anweisung muß sich durch ihre logische Dateinummer auf eine OPEN-Anweisung beziehen, in der die Geräteadresse 4 und die Sekundäradresse 1 angegeben worden war. Die einzelnen Zeichenketten in der Parameterliste einer PRINT#-Anweisung sind mit dem Steuerkode CHR$ (29) voneinander zu trennen. Dieser Steuerkode kann bei der Programmeingabe auch mit der Taste KURSOR NACH RECHTS gebildet werden (siehe Tabelle 6-4). Zeichenketten werden linksbündig in dem durch eine Formatangabe definierten Zeichenfeld dargestellt; nicht belegte Zeichenpositionen am Ende der Zeichenkette werden durch Leerstellen aufgefüllt, führende Leerstellen abgeschnitten.

Hier ein Beispiel für die beiden eben besprochenen PRINT#-Anweisungen:

```
100 PRINT#X,"AAAAAAAAAA     AAAAAAAAAAAA"
110 PRINT#Y,M$CHR$(29)N$
```

Die PRINT#X-Anweisung in Zeile 100 vereinbart das Druckformat, in diesem Fall ein zehn-stelliges und ein zwölf-stelliges Zeichenfeld mit einem Abstand von fünf Leerstellen.

Die PRINT#Y-Anweisung in Zeile 110 enthält die beiden auszudruckenden Zeichenketten in Form der Variablen M$ und N$, getrennt durch das vorgeschriebene Symbol CHR$ (29). Beachten Sie das Fehlen von Kommas in der Parameterliste der PRINT#X-Anweisung; Sie könnten aber Kommas verwenden, wenn Sie wünschen, und die PRINT#X-Anweisung wie folgt schreiben:

```
PRINT#Y,M$,CHR$(29),N$
```

Die Variablen M$ und N$ könnten auch durch die tatsächlichen Zeichenketten ersetzt werden, mit oder ohne Kommas zwischen den Variablen und dem Trennsymbol CHR$ (29). Folgendes Beispiel soll diesen Fall illustrieren:

```
PRINT#Y,"EINS"CHR$(29)"ZWEI"
```

Eine Vorstellung von formatiert ausgedruckten Zeichenketten kann das nachfolgend aufgelistete Programm PRINT.FORM.ZEICHK. geben, das eine Abwandlung des Programms NUM.FORM.PRINT. darstellt:

PRINT.FORM.ZEICHK.

```
10 REM PROGRAMM ** ZEICHENKETTEN-AUSDRUCK **
20 REM ZUR DEMONSTRATION FORMATIERTER ZEICHENKETTEN-AUSDRUCKE
30 DATA "HANS MUELLER","KAISERSTR.50","8 MUENCHEN 2","BRD"
35 DATA "GESCH: 580 3571","PRIV: 722 2678","MITGLIED","A9"
70 OPEN 1,4,1 : REM  DATEN -AUSGABE UEBER KANAL 1
80 OPEN 2,4,2 : REM  FORMAT-AUSGABE UEBER KANAL 2
```

```
90 REM AUSGABE DES FORMATS:
100 PRINT#2,"AAAAAAAAAA      AAAAAAAAAAAA"
105 ZW$=CHR$(29)
109 REM AUSGABE DER DATEN:
110 FOR I=1 TO 4
120 READ M$,N$
130 PRINT#1,M$,ZW$,N$
140 NEXT I
150 CLOSE 1
160 CLOSE 2
170 STOP

HANS MUELL      KAISERSTR.50
8 MUENCHEN      BRD
GESCH: 580      PRIV: 722 26
MITGLIED        A9
```

Die PRINT#X-Anweisung erscheint in Zeile 100, hier mit der logischen Dateinummer X = 2 in Entsprechung zur OPEN-Anweisung in Zeile 80. Die PRINT#Y-Anweisung erscheint in Zeile 130 mit der logischen Dateinummer Y = 1 als Bezug zur OPEN-Anweisung in Zeile 70. Das Trennsymbol CHR$ (29) in der PRINT#-1-Anweisung in Zeile 130 wurde stellvertretend dargestellt durch ZW$ dessen Zuweisung zu CHR$ (29) in Zeile 105 steht.

Die ursprünglich 8 Zahlen in der DATA-Anweisung des Programms NUM. FORM.PRINT sind im vorliegenden Programm durch 8 Zeichenketten in den DATA-Anweisungen in Zeilen 30 und 35 ersetzt worden.

Am ersten Ausdruck dieses Programms kann man feststellen, daß einige der in DATA angegebenen Zeichenfelder abgehackt wurden. Die Formatanweisungen in Zeile 100 müssen durch Anfügen weiterer Buchstaben A an die Länge der Zeichenketten angepaßt werden. Beachten Sie auch, daß alle Zeichenketten linksbündig ausgedruckt wurden. Leerstellen vor ausgedruckten Zeichenketten werden mit der Funktion CHR$ (160) dargestellt, die auch den beiden Tasteneingaben SHIFT und Leertaste entspricht. Als Beispiel werden wir führende Leerstellen vor eine der Zeichenketten setzen; wir wählen das letzte Zeichenfeld A9. Ändern Sie die DATA-Anweisung in Zeile 35 wie folgt:

```
35 DATA "GESCH: 580 3571","PRIV: 722 2678","MITGLIED","  A9"
```

Leertaste zweimal drücken

Ein erneuter Programmlauf führt zu keiner Änderung des Ausdrucks. Die beiden Leerstellen vor A9 wurden ignoriert. Geben Sie jetzt die Leerstellen ein, indem Sie die Umschalttaste (SHIFT) gedrückt halten, während Sie mit der Leertaste zwei Leerstellen vor A9 eingeben. Diesmal erscheinen zwei Leerstellen vor A9 im Druckbild.

Die Rechtsverschiebung von Zeichenketten innerhalb des Zeichenfeldes einer Formatanweisung ist über eine Verkettung mit den oben genannten Leerstellen möglich. Zur Veranschaulichung dieses Verfahrens haben wir das Programm PRINT. FORM-ZEICHK. wie folgt abgewandelt:

PRINT.FORM.ZEICHK. 1

```
10 REM PROGRAMM ** ZEICHENKETTEN-AUSDRUCK (1) **
20 REM ZUR DEMONSTRATION FORMATIERTER ZEICHENKETTEN-AUSDRUCKE
30 DATA "HANS MUELLER","KAISERSTR.50","8 MUENCHEN 2","BRD"
35 DATA "GESCH: 580 3571","PRIV: 722 2678","MITGLIED","A9"
70 OPEN 1,4,1 : REM  DATEN -AUSGABE UEBER KANAL 1
80 OPEN 2,4,2 : REM  FORMAT-AUSGABE UEBER KANAL 2
```

```
90 REM AUSGABE DES FORMATS:
100 PRINT#2,"AAAAAAAAAA      AAAAAAAAAAAA"
105 ZW$=CHR$(29)
106 LR$="              " : REM 12 SHIFT-LEERTASTEN
109 REM AUSGABE DER DATEN:
110 FOR I=1 TO 4
120 READ M$,N$
125 IF LEN(M$)<10 THEN M$=LEFT$(LR$,(10-LEN(M$)))+M$
126 IF LEN(N$)<12 THEN N$=LEFT$(LR$,(12-LEN(N$)))+N$
130 PRINT#1,M$ZW$N$
140 NEXT I
150 CLOSE 1
160 CLOSE 2
170 STOP


HANS MUELL      KAISERSTR.50
8 MUENCHEN            BRD
GESCH: 580      PRIV: 722 26
  MITGLIED            A9
```

Vor einer Rechtsverschiebung wird zunächst in Zeilen 125 und 126 die Länge der Zeichenketten mit den Feldlängen der Formatanweisungen verglichen. Kürzere Zeichenketten werden mit SHIFT + Leertaste-Symbolen auf die Feldlänge der Formatanweisung ergänzt. Hierzu dient die Variable LR$ in Zeile 106 als Reservoir für Leerstellen, die mit SHIFT-Taste und Leertaste gebildet wurden. Die LEFT$-Funktion in Zeilen 125 und 126 verkettet führende Leerstellen entsprechend den Längendifferenzen von Zeichenkette und Zeichenfeld mit den zugehörigen Zeichenketten.

Wir werden schließlich das Programm PRINT.FORM.ZEICHK. derart abändern, daß es die Adreßangabe in einer vernünftigen Form ausdruckt. Zum Beispiel können die ersten fünf Zeichenketten aus der DATA-Anweisung in Zeile 30 in vertikaler Anordnung ausgedruckt werden, während die drei restlichen Zeichenketten auf einer Zeile stehen. Das unten aufgelistete Programm PRINT.FORM.ZEICHK. 2 erzeugt ein derartiges Druckbild:

PRINT.FORM.ZEICHK. 2

```
10 REM PROGRAMM ** ZEICHENKETTEN-AUSDRUCK (2) **
20 REM ZUR DEMONSTRATION FORMATIERTER ZEICHENKETTEN-AUSDRUCKE
30 DATA "HANS MUELLER","KAISERSTR.50","8 MUENCHEN 2","BRD"
35 DATA "49-89-580 3571","MITGLIED","A9"
70 OPEN 1,4,1 : REM  DATEN -AUSGABE UEBER KANAL 1
80 OPEN 2,4,2 : REM  FORMAT-AUSGABE UEBER KANAL 2
100 REM LESEN DER DATEN:
110 FOR I=1 TO 7
120 READ M$(I)
140 NEXT I
145 REM AUSGABE DES FORMATS FUER M$(1) BIS M$(4):
150 PRINT#2,"AAAAAAAAAAAAAAAA"
160 FOR I=1 TO 4
170 PRINT#1,M$(I)
180 NEXT I
185 REM AUSGABE DES FORMATS FUER M$(5) BIS M$(7):
190 PRINT#2,"AAAAAAAAAAAAAAA   AAAAAAAA   AA"
195 ZW$=CHR$(29)
200 PRINT#1,M$(5)ZW$M$(6)ZW$M$(7)
210 CLOSE 1
220 CLOSE 2
230 STOP
```

```
HANS MUELLER
KAISERSTR.50
8 MUENCHEN 2
BRD
49-89-580 3571    MITGLIED    A9
```

Alle acht Zeichenketten der DATA-Anweisungen wurden zunächst mit einer FOR-NEXT-Schleife (Zeilen 110 bis 140) in die Tabelle M$ (I) gelesen, ehe ein Ausdruck erfolgt. Fünf Zeichenketten werden dann in vertikaler Anordnung mit der FOR-NEXT-Schleife in Zeilen 160 bis 180 ausgedruckt, wofür der Drucker die Formatanweisung von Zeile 150 verwendet. Die restlichen drei Zeichenketten werden mit einer einzelnen PRINT#-Anweisung in Zeile 200 ausgedruckt, nachdem der Drucker in Zeile 190 die Formatanweisung für diese Zeichenketten erfahren hat.

FORMATIERTER AUSDRUCK GEMISCHTER DATEN

Es sind auch gemischte Formatangaben für numerische Daten und Zeichenketten möglich. Derartige gemischte Formatangaben sind in dem unten aufgelisteten Programm PRINT.FORM.ZEICHK. 3 enthalten:

PRINT.FORM.ZEICHK. 3

```
10 REM PROGRAMM ** ZEICHENKETTEN-AUSDRUCK (3) **
20 REM ZUR DEMONSTRATION FORMATIERTER ZEICHENKETTEN-AUSDRUCKE
30 DATA "HANS MUELLER","KAISERSTR.50","8 MUENCHEN 2","BRD"
35 DATA "49-89-580 3571","MITGLIED","A9"
70 OPEN 1,4,1 : REM  DATEN -AUSGABE UEBER KANAL 1
80 OPEN 2,4,2 : REM  FORMAT-AUSGABE UEBER KANAL 2
100 REM LESEN DER DATEN:
110 FOR I=1 TO 7
120 READ M$(I)
140 NEXT I
145 REM AUSGABE DES FORMATS FUER M$(1) BIS M$(4):
150 PRINT#2,"99  AAAAAAAAAAAAAAA"
160 FOR I=1 TO 4
170 PRINT#1,I,M$(I)
180 NEXT I
185 REM AUSGABE DES FORMATS FUER M$(5) BIS M$(7):
190 PRINT#2,"99   AAAAAAAAAAAAAA   AAAAAAAA   AA"
195 ZW$=CHR$(29)
200 PRINT#1,5,M$(5)ZW$M$(6)ZW$M$(7)
210 CLOSE 1
220 CLOSE 2
230 STOP
```

```
1   HANS MUELLER
2   KAISERSTR.50
3   8 MUENCHEN 2
4   BRD
5    49-89-580 3571    MITGLIED    A9
```

Dieses Programm entstand durch kleinere Abänderungen in Programm PRINT.FORM.ZEICHK. 1. Die PRINT#-Anweisungen in Zeilen 150 und 190 enthalten zusätzlich die Formatangabe für eine Zeilenzahl vor den ausgedruckten Zeichenketten. Der Ausdruck dieser Zeilenzahlen erfolgt mit den PRINT#-Anweisungen in Zeilen 170 und 200.

Ein weiteres Programm, PRINT.DATUM, liest die Tastenfeld-Eingabe eines Datums in Form von drei numerischen Variablen für Tag, Monat und Jahr. Jedes Datum wird mit einem Punkt als Trennsymbol zwischen Tag und Monat und Monat und Jahr ausgedruckt. Hier die Auflistung des Programms und das Beispiel einiger Programmläufe:

PRINT.DATUM

```
10 REM PROGRAMM ** DATUM AUSDRUCKEN **
20 OPEN 1,4,1 : REM DATEN -AUSGABE UEBER KANAL 1
30 OPEN 2,4,2 : REM FORMAT-AUSGABE UEBER KANAL 2
40 PRINT"■"
50 INPUT "TAG    :";T
60 INPUT "MONAT  :";M
70 INPUT "JAHR   :";J
80 PRINT#2,"AAAAAA     99A99A99"
90 ZW$=CHR$(29)
100 PRINT#1,"DATUM:"ZW$,T,"."ZW$,M,"."ZW$,J
110 PRINT "WEITERE DATUM-EINGABE? J ODER N EINGEBEN"
120 GET JN$ : IF JN$="" THEN 120
130 IF JN$= "N" THEN PRINT JN$ : STOP
140 IF JN$<>"J" THEN 120
150 GOTO 40
```

```
DATUM:     15. 4.81
DATUM:     10.10.81
DATUM:      1. 1.82
```

Das Programm enthält keine Prüfungen auf gültige Zahleneingaben für Tag, Monat und Jahr, da unser Interesse der Formatierung galt. Noch offensichtlicher ist die Nützlichkeit formatierter Ausdrucke im nachfolgenden Beispiel.

AUSDRUCK VON SYMBOLEN IN FORMATANWEISUNGEN

In Formatanweisungen für Drucker können auch Symbole aufgenommen werden, die genau so ausgedruckt werden, wie sie in der Formatanweisung stehen. Derartige Symbole werden also nicht mit einer PRINT#-Anweisung für Druckdaten, sondern zusammen mit der Formatanweisung an den Drucker übertragen. Symbole in einer Formatanweisung müssen durch ein voranstehendes Zeichen für Negativschrift (OFF RVS-Taste) gekennzeichnet werden. Ein unmittelbar nach dem Negativschrift-Zeichen folgendes Symbol wird normal ausgedruckt. Folglich können mit einer Formatanweisung keine Symbole in Negativschrift ausgedruckt werden.

Das folgende Programm PRINT.DATUM.1 gibt ein einfaches Beispiel für die Anwendung. Als Symbol wurde ein Sternzeichen statt des Punktes im vorigen Programm aufgenommen. Hier das Programm mit seinen Änderungen und einigen Ausdrucken:

PRINT.DATUM. 1

```
10 REM PROGRAMM ** DATUM AUSDRUCKEN (1) **
20 OPEN 1,4,1 : REM DATEN -AUSGABE UEBER KANAL 1
30 OPEN 2,4,2 : REM FORMAT-AUSGABE UEBER KANAL 2
40 PRINT"■"
50 INPUT "TAG    :";T
60 INPUT "MONAT  :";M
70 INPUT "JAHR   :";J
80 PRINT#2,"AAAAAA     99■99■99"
```

```
90 ZW$=CHR$(29)
100 PRINT#1,"DATUM:"ZW$,T,M,J
110 PRINT "WEITERE DATUM-EINGABE? J ODER N EINGEBEN"
120 GET JN$ : IF JN$="" THEN 120
130 IF JN$= "N" THEN PRINT JN$ : STOP
140 IF JN$<>"J" THEN 120
150 GOTO 40


DATUM:     12* 4*81
DATUM:     24*12*81
DATUM:      1* 1*82
```

STEUERZEICHEN FÜR DRUCKER

Es steht eine Reihe von Steuerzeichen für Drucker zur Verfügung, die in die auszudruckenden Daten eingefügt werden und das Druckbild zu verändern gestatten. Tabelle 6-4 enthält eine Zusammenfassung derartiger Steuerzeichen.

Steuerzeichen werden wie Daten mit einer Sekundäradresse 0 oder 1 zum Drucker übertragen und nicht als Teil einer Formatanweisung, das heißt mit einer Sekundäradresse 2 übertragen.

Steuerzeichen können bei formatierten oder unformatierten Ausdrucken gesendet werden. Lediglich die ersten beiden Steuerzeichen in Tabelle 6-4, CHR$ (29) und CHR$ (160), müssen im Zusammenhang formatierter Ausdrucke verwendet werden. Sie werden in unformatierten Ausdrucken ignoriert.

BREITSCHRIFT : CHR$ (1).

Der Schreibkopf in CBM-Druckern verwendet eine Punkt-Matrix zum Ausdrucken von Zeichen. Die Höhe dieser Matrix ist 7 Punkte und ihre Breite 6 Punkte (CBM 3022, CBM 3023, CBM 4022) bzw. 4 Punkte (CBM 8024). Durch Einfügen des Zeichens CHR$ (1) in die Liste auszudruckender Daten einer PRINT#-Anweisung werden alle Zeichen nach CHR$ (1) mit doppelter Breite ausgedruckt. Die Druckmatrix ist dann 7 Punkte hoch und 12 Punkte breit (7 Punkte breit bei CBM 8024).

Die Wirkung des Steuerzeichens CHR$ (1) ist kumulativ: Jedes CHR$ (1) verdoppelt die zuvor vorhandene Zeichenbreite. Zwei Steuerzeichen CHR$ (1) in einer Liste auszudruckender Daten vervierfachen die Zeichenbreite, z.B. auf 7 x 24 Punkte. Folgt ein drittes Steuerzeichen CHR$ (1), dann werden Zeichen in achtfacher Breite dargestellt, d.h. in einer Matrix von 7 x 48 Punkten.

Zur Veranschaulichung der Breitschrift haben wir im Programm PRINT. FORM.ZEICHK. 1 die Zeilen 165 und 196 zugefügt, um in den jeweils nachfolgenden PRINT#-Anweisungen Ausdrucke in Breitschrift zu erhalten:

```
165 M$(I)=CHR$(1)+M$(I)
170 PRINT#1,M$(I)
 .
 .
196 BR$=CHR$(1)
200 PRINT#1,BR$M$(5) ZW$ BR$M$(6) ZW$ BR$M$(7)
```

Das Programm druckt jetzt die ersten vier Zeilen mit doppelter Zeichenbreite aus. In der fünften Zeile sehen wir die kumulative Wirkung von CHR$ (1): Die Telefonnummer wird mit doppelter Zeichenbreite ausgedruckt, das Wort "Mitglied" in vierfacher Zeichenbreite und der Kode "A9" in achtfacher Breite. Hier der Ausdruck:

```
HANS MUELLER
KAISERSTR.50
8 MUENCHEN 2
BRD
49-89-580 3571    MITGLIED A9
```

Wie wurde das erreicht?
Zeile 165 stellt jeder Zeichenketten-Variablen M$ (I) das Steuerzeichen CHR$ (1) voran, so daß die nachfolgende PRINT#-Anweisung die Zeichenketten M$ (1) bis M$ (4) in doppelter Schriftbreite darstellt. In Zeile 196 wird das Sonderzeichen CHR$ (1) der Variablen BR$ zugewiesen, um in dieser Form in der folgenden PRINT#-Anweisung verwendet zu werden. Nach jedem BR$ in der Parameterliste der PRINT#-Anweisung in Zeile 200 wird die Zeichenbreite verdoppelt. Man sieht daher in der fünften Zeile des Ausdrucks die kumulative Wirkung von CHR$ (1).

Das Steuerzeichen CHR$ (1) muß nicht wie in Zeile 165 durch Verkettung vor eine Zeichenkette gestellt werden. Sie können CHR$ (1) auch einfach in die Parameterliste einer PRINT#-Anweisung einfügen, wie in Zeile 200 geschehen. Jedoch darf CHR$ (1) nicht durch Kommas abgetrennt werden.

Nach dem Steuerzeichen für Breitschrift, CHR$ (1), können auch numerische Variable in die Parameterliste einer PRINT#-Anweisung aufgenommen werden, die jedoch durch Kommas von den übrigen Variablen in der Parameterliste zu trennen sind. Zur Veranschaulichung haben wir die Zeilen 190 bis 200 im eben benutzten Programm PRINT.FORM.ZEICHK. 2 wie folgt verändert:

```
190 PRINT#2,"AAAAAAAAAAAAAA   99999999   AA"
195 ZW$=CHR$(29)
196 BR$=CHR$(1)
197 N=12345678
200 PRINT#1,BR$M$(5)ZW$,N,M$(7)
```

Die Formatanweisung in Zeile 190 definiert jetzt zwei Felder für Zeichenketten (A) und ein Feld für Zahlen (9). In dieses Zahlenfeld wird die Zahl aus Zeile 196, N, eingetragen. In Zeile 200 enthält die Parameterliste der PRINT#-Anweisung an erster Stelle das Steuerzeichen CHR$ (1), wodurch alle Variable in der Parameterliste in Breitschrift ausgedruckt werden. Beachten Sie, daß nur die numerische Variable durch Kommas abgetrennt ist, während zwischen den Zeichenketten-Variablen keine Kommas verwendet werden.

Diese Anwendungsregeln sind sehr eigentümlich für die Programmierung von Ausdrucken in Breitschrift und müssen sorgfältig beachtet werden. Hier das Ergebnis eines Programmlaufs von PRINT.FORM.ZEICHK. 2 nach Änderung der Zeilen 190 bis 200:

```
HANS MUELLER
KAISERSTR.50
8 MUENCHEN 2
BRD
49-89-580 3571    12345678    A9
```

Das Steuersignal für Breitschrift, CHR$ (1), kann mit dem Steuerzeichen CHR$ (129) wieder gelöscht werden. Nachfolgende Zeichen werden wieder in normaler Größe ausgedruckt, bis ein nächstes CHR$ (1) in der Parameterliste auftritt.

AUSDRUCKE IN NEGATIVSCHRIFT

Variable in der Parameterliste einer PRINT#-Anweisung können mit den Tasteneingaben OFF RVS (Negativschrift EIN) und SHIFT+OFF RVS (Negativschrift AUS) in Negativschrift ausgedruckt werden. Zur Schonung des Druckkopfes sollte man jedoch nicht ausgiebig von dieser Möglichkeit Gebrauch machen.

AUSDRUCK VON STEUERZEICHEN

Steuerzeichen der Form CHR$ ( ) in einer PRINT#-Anweisung werden auf dem Drucker durch graphische Symbole wiedergegeben, wenn zuvor das Steuerzeichen für ein Anführungszeichen, CHR$ (34), eingegeben wurde.

Von dieser Möglichkeit wird lediglich bei Auflistungen von Programmen Gebrauch gemacht, deren Steuerzeichen üblicherweise nicht dargestellt werden.

FORMATIERUNG VON DRUCKSEITEN

ZEILENZAHL JE SEITE : CHR$ (147)

Bei Fehlen einer anderen Anweisung erfolgen Ausdrucke von CBM-Druckern ohne Einhaltung eines Seitenformats. Die Gestaltung des Seitenformats ist durch Ausgabe des Steuerzeichens CHR$ (147) an den Drucker möglich. Nach Empfang dieses Steuerzeichens nimmt der Drucker ein Seitenformat mit 66 Zeilen je Seite an; hiervon werden 60 Zeilen bedruckt, dann erfolgt ein Sprung von sechs Zeilen und die nächsten 60 Zeilen werden gedruckt.

Das nachfolgende einfache Programm aktiviert zunächst die Möglichkeit zur Seitenformatierung im Drucker und druckt dann 100 Zeilen mit durchgehender Zeilennumerierung und dem Text ABCDEFG. Am Ausdruck dieses Programms können Sie eine erste Möglichkeit der Seitenformatierung sehen:

ZEILENZAHL = 66

```
10 REM PROGRAMM ** 66 ZEILEN JE SEITE **
20 OPEN 1,4 : REM  OEFFNEN OHNE FORMATANGABE
30 REM  STEUERUNG DER ZEILEN/SEITE EIN:
40 PRINT#1,CHR$(147)
45 REM  DATEN-AUSGABE:
50 FOR I=1 TO 100
60 PRINT#1,I,"ABCDEFG"
70 NEXT I
80 CLOSE 1
90 STOP
```

Die Zeilenzahl je Seite kann aber auch durch Sie bestimmt werden, sobald die Seitenformatierung mit CHR$ (147) im Drucker aktiviert wurde. Die gewünschte Zeilenzahl wird mit einer PRINT#-Anweisung dem Drucker mitgeteilt, wofür eine logische Datei mit der Geräteadresse 4 und der Sekundäradresse 3 zu öffnen ist. Der Drucker bedruckt dann die angegebene Zahl von Zeilen, gefolgt von einem Zeilenvorschub von 6 Zeilen an den Anfang der nächsten Seite. Das hier aufgelistete Programm sieht 25 Zeilen je Seite vor:

ZEILENZAHL = 25

```
10 REM PROGRAMM ** 25 ZEILEN JE SEITE **
20 OPEN 1,4    : REM  KANAL FUER DATEN -UEBERTRAGUNG
25 OPEN 3,4,3  : REM  KANAL FUER FORMAT-ANGABE(ZEILEN/SEITE)
30 REM  STEUERUNG DER SEITENFORMATIERUNG "EIN":
40 PRINT#1,CHR$(147)
44 REM  ANGABE DER ZEILEN/SEITE(= 25):
45 PRINT#3,25
49 REM  DATEN-AUSGABE:
50 FOR I=1 TO 100
60 PRINT#1,I,"ABCDEFG"
70 NEXT I
80 CLOSE 1
85 CLOSE 3
90 STOP
```

Die PRINT#-Anweisung in Zeile 45 teilt die Zeilenzahl 25 je Seite mit. Die zugehörige logische Datei wurde in Zeile 25 geöffnet.

Solange die logische Datei für die Übermittlung der Zeilenzahl je Seite geöffnet ist, kann die Zeilenzahl je Seite im Programmverlauf geändert werden. Eine neue Angabe der Zeilenzahl wird mit Beginn der nächsten Seite wirksam; der laufende Seitenausdruck wird noch mit der alten Angabe beendet. Hier das Beispiel einer Programmzeile, mit der im Programmlauf die Zeilenzahl geändert wird:

```
55 IF I=23 THEN PRINT#3,10
```

Lassen Sie das Programm zweimal ablaufen. Beim ersten Mal wird eine 25-zeilige Seite gedruckt, gefolgt von einer Anzahl 10-zeiliger Seiten. Beim zweiten Programmlauf ereignet sich jedoch etwas seltsames: Zunächst wird eine 10-zeilige Seite ausgedruckt, gefolgt von einer 25-zeiligen Seite, und dann eine Reihe von 10-zeiligen Seitenausdrucken. Der Drucker hielt die zuletzt vereinbarte Zeilenzahl 10 noch gespeichert und benutzte sie für die erste Seite im neuen Programmlauf.

SEITENVORSCHUB : CHR$ (19)

Der Drucker führt bei Empfang des Steuerzeichens CHR$ (19) einen Seitenvorschub aus, in dem die restlichen Zeilen des laufenden Seitenausdrucks übersprungen werden. Im Drucker muß hierfür die Seitenformatierung durch CHR$ (147) noch wirksam sein. Der Drucker setzt seine Ausdrucke am neuen Seitenbeginn fort. Wird eine Seite nicht bis zur letzten Zeile gedruckt (was eher die Regel als die Ausnahme ist), dann sollten Sie den Text jeder Seite mit dem Zeichen CHR$ (19) abschließen, so daß sich der Drucker auf den nächsten Seitenanfang einstellen kann. Die Zählung restlicher Zeilen auf einer Seite und ihr Überspringen ist also nicht erforderlich.

VERÄNDERN DES ZEILENABSTANDS

Die Drucker CBM 4022 und CBM 3022 ermöglichen eine programmierte Änderung des Zeilenabstands. Jeder Papiervorschub von einem Zoll (= 2,54 cm) wird bei diesen Druckern in 144 Schritte unterteilt. Jede Zeile wird normalerweise aus 24 Schritten gebildet. Das ergibt 6 Zeilen je 1 Zoll Papiervorschub. Die Modelle CBM 4022 und CBM 3022 ermöglichen die Programmierung der Zahl der Zeilen je 1 Zoll Papiervorschub. Hierfür muß eine logische Datei mit der Geräteadresse 4 und der Sekundäradresse 6 geöffnet werden. Dann wird mit einer PRINT#-Anweisung die Zahl der Schritte je Zeile als Argument in der Funktion CHR$ dem Drucker mitgeteilt.

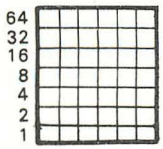
Typische Zeilenzahlen sind 6 oder 8 Zeilen je Zoll; die Zahl der Schritte je Zoll ist hierfür 144/6 bzw. 144/8, d.h. 24 bzw. 18 Schritte. Hier die Anweisungen für die Programmierung von 8 Zeilen je Zoll:

```
10 OPEN 6,4,6
20 PRINT#6,CHR$(18)
```

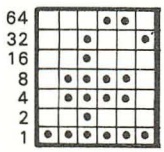
Fügen Sie diese Zeilen in das Programm ZEILENZAHL = 25 ein. Der Programmlauf erzeugt auf einem Drucker CBM 4022 (bzw. CBM 3022) Druckzeilen ohne Zeilenabstand. Da sich die vertikale Höhe der Druckzeichen bei Änderung der Zeilenzahl je Zoll nicht ändert, können Sie überlappende Druckzeilen oder Druckzeilen mit großen Zeilenabständen ausgeben.

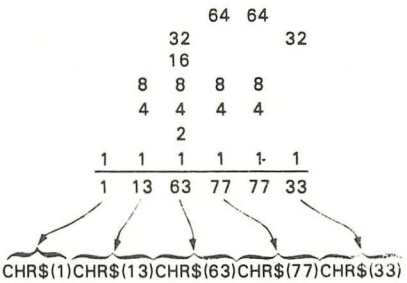
SELBSTENTWORFENE DRUCKZEICHEN

Die Drucker CBM 3022, CBM 3023, und CBM 4022 geben Ihnen die Möglichkeit, Druckzeichen selbst zu entwerfen und vom Drucker ausgeben zu lassen. Zur Zeichenbildung benutzen diese Drucker eine Matrix von 6 x 7 Punkten. Verwenden Sie das Schema dieser Matrix für den Entwurf Ihres eigenen Druckzeichens:



Jede Zeile dieser Punkt-Matrix ist numeriert. Die Zeilennummern verdoppeln sich von unten nach oben. Die Anwendung dieser Zeilenzahlen bei der Verschlüsselung Ihres selbstentworfenen Druckzeichens zeigt das folgende Beispiel anhand des Pfund-Zeichens der englischen Währung:





Jedes Druckzeichen wird mit 6 Zahlen verschlüsselt, deren Bildung leicht aus unserem Beispiel erkennbar ist. Als nächstes werden diese 6 Zahlen in eine Zeichenkette umgewandelt, die aus 6 Zeichen der Form CHR$ ( ) bestehen. Argument dieser CHR$-Funktionen sind die bei der Verschlüsselung entstandenen 6 Zahlen. Diese Zeichenkette wird dann mit einer PRINT#-Anweisung zum Drucker übertragen, der hierdurch das selbstentworfene Druckzeichen speichert. Zur Übertragung des Druckzeichens zum Drucker muß eine logische Datei mit der Geräteadresse 4 und der Sekundäradresse 5 geöffnet werden. Der eigentliche Ausdruck des Zeichens wird von

jeder PRINT#-Anweisung ausgelöst, die das Steuerzeichen CHR$ (254) enthält.

Das folgende Programm stellt eine Spalte mit 10 englischen Pfund-Zeichen dar:

PFUND-SYMBOL (1)

```
10 REM PROGRAMM     ** ENGLISCHES PFUNDZEICHEN **
20 REM DEMONSTRIERT BILDUNG VON SONDERZEICHEN IM DRUCKER
30 DATA 1,13,63,77,77,33 : REM KODEZAHLEN DES SONDERZEICHENS
35 EP$=""
40 OPEN 1,4      : REM DATENKANAL ZUM DRUCKER OEFFNEN
50 OPEN 5,4,5    : REM DATEI FUER SONDERZEICHEN OEFFNEN
60 FOR I=1 TO 6 : REM SONDERZEICHEN IN EP$ BILDEN
70 READ EP
80 EP$=EP$+CHR$(EP)
90 NEXT I
95 PRINT#5,EP$   : REM SONDERZEICHEN ZUM DRUCKER SENDEN
100 FOR I=1 TO 10
110 PRINT#1,CHR$(254)
120 NEXT I
130 CLOSE 1
140 CLOSE 5
150 STOP
```

Lassen Sie uns untersuchen, wie das Pfund-Zeichen gebildet und ausgedruckt wird.

Die DATA-Anweisung in Zeile 30 enthält 6 Zahlen, die bei der Verschlüsselung des Pfund-Zeichens entstanden sind, wie wir oben gezeigt haben. Diese 6 Zahlen werden in einer FOR-NEXT-Schleife (Zeilen 60 - 90) gelesen, um die Zeichenkette EP$ zu bilden, mit der das Pfund-Zeichen dem Drucker übermittelt wird. EP$ wird in Zeile 80 schrittweise gebildet und in Zeile 95 mit der PRINT#5-Anweisung zum Drucker übertragen. Die logische Datei 5 wurde in Zeile 50 mit einer Geräteadresse 4 und Sekundäradresse 5 geöffnet.

Nach Ausführung der PRINT#-Anweisung in Zeile 95 hat der Drucker das Pfund-Symbol gespeichert, dessen Ausdruck durch eine PRINT#-Anweisung mit dem Parameter CHR$ (254) ausgelöst wird.

Beachten Sie, daß ein CBM-Drucker jeweils nur **ein** selbstentworfenes Druckzeichen speichern kann. Sie können selbstverständlich weitere Druckzeichen entwerfen und der Reihe nach verschlüsselt dem Drucker übermitteln.

ANWENDUNG SELBSTENTWORFENER DRUCKZEICHEN

Wir werden das eben besprochene Druckzeichen im folgenden Programm benutzen, um Zahlen als englische Pfund auszudrucken:

PFUND-SYMBOL (2)

```
10 REM PROGRAMM  ** PFUND-SYMBOL (2) **
20 REM ZAHLEN AUSGEDRUCKT ALS BRITISCHE PFUND
30 REM  PFUND-ZEICHEN BILDEN:
40 DATA 1,13,63,77,77,33
50 OPEN 1,4,5
60 EP$=""
70 FOR I=1 TO 6
80 READ EP
90 EP$=EP$+CHR$(EP)
100 NEXT I
105 REM  PFUND-ZEICHEN IM DRUCKER SPEICHERN:
110 PRINT#1,EP$
115 REM  AUSDRUCK FORMATIEREN:
```

```
120 OPEN  2,4,2 : CMD 2
130 PRINT#2,"AAAAAAA A 999999.99-"
140 REM  LESEN UND AUSDRUCKEN DES PFUND-BETRAGS:
150 OPEN  3,4,1
160 INPUT "BETRAG EINGEBEN:";N
170 PRINT#3,"BETRAG="CHR$(29)CHR$(254)CHR$(29)N
180 CLOSE 1
190 CLOSE 2
200 CLOSE 3
210 STOP
```

```
BETRAG=  £ 123456.78

BETRAG=  £ 123456.78-
```

Die OPEN-Anweisungen in Zeilen 120 und 130 öffnen die logischen Dateien 1 und 2 für die Ausgabe der Formatanweisungen (Sekundäradresse 2) und der formatiert zu druckenden Daten (Sekundäradresse 1). Die Formatanweisung ist in der PRINT#-Anweisung in Zeile 150 enthalten.

Die INPUT-Anweisung in Zeile 160 bittet Sie um Eingabe einer Zahl, die der numerischen Variablen N zugewiesen und in der PRINT#-Anweisung in Zeile 170 ausgedruckt wird.

Die Parameterliste der PRINT#-Anweisung in Zeile 170 enthält als erste Zeichenkette "BETRAG = ", deren Ausdruck mit der Formatanweisung "AAAAAAA" vorbereitet wurde. Danach folgt das Sprungzeichen CHR$ (29), mit dem jede Zeichenkette abgeschlossen werden muß. Die folgende Zeichenkette CHR$ (254) löst den Ausdruck des Pfund-Zeichens in das nächste formatierte Feld aus. Nach einem weiteren Sprungzeichen CHR$ (29) folgt schließlich, getrennt durch ein Komma, die numerische Variable N zur Darstellung des Pfund-Betrags.

FEHLERMELDUNGEN DES DRUCKERS

Wenn Sie Schwierigkeiten mit den Ausdrucken Ihres Druckers haben, sollten Sie eine logische Datei mit der Geräteadresse 4 und der Sekundäradresse 4 öffnen. Hierdurch erhält der Drucker die Möglichkeit zu ausführlichen Fehlermeldungen, falls er erkennbare Fehler in den Format-Angaben Ihres Ausdrucks erkennt. Diese Fehlermeldungen werden automatisch ausgedruckt.

Fehlermeldungen des Druckers werden gewöhnlich während einer Programmentwicklung benutzt und nicht in der endgültigen Fassung eines Programms belassen.

Laden Sie noch einmal das Programm PRINT.FORM.ZEICHK.2 und fügen folgende Zeile hinzu:

```
85 OPEN 4,4,4
```

Ein Fehler in der Formatanweisung wurde mit folgender Fehlermeldung gezeigt:

```
HANS MUELLER
KAISERSTR.50
8 MUENCHEN 2
BRD
49-89-
AAAAAA4AAAAAAA    AAAA5AAA    AA
      ↑
*PE:F*
MITGLIED A9
```

KAPITEL 7

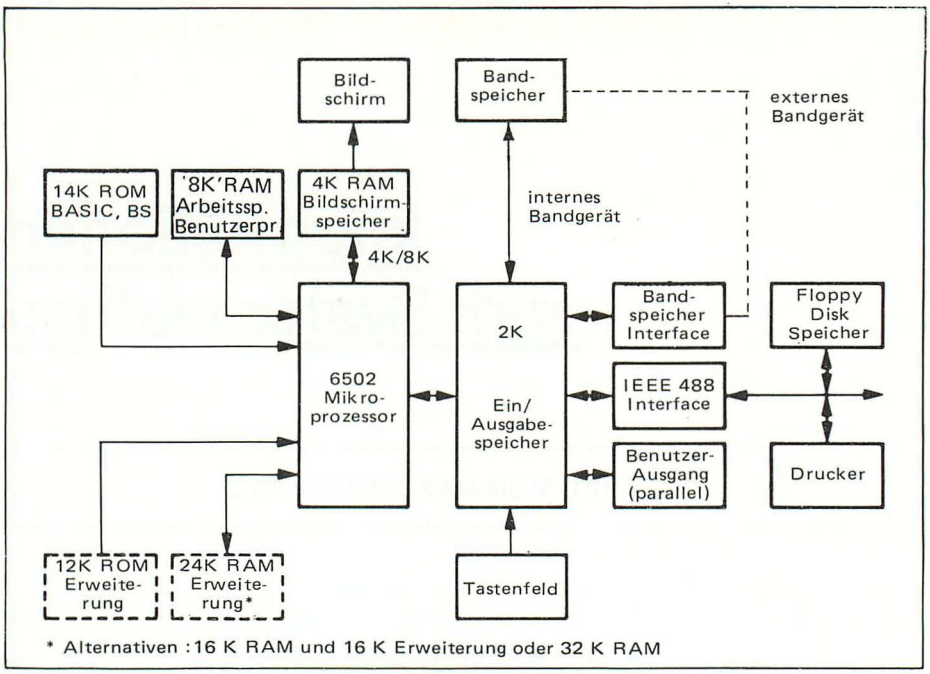
Informationen zum Systemaufbau

AUFBAU VON CBM-COMPUTERN

CBM-Computer arbeiten mit dem Mikroprozessor 6502, der das Herz des Systemaufbaus von CBM-Computern bildet (Figur 7-1). Der Datenverkehr mit externen Geräten führt über drei Ein/Ausgabe-Tore, an denen auch die Umsetzung zwischen Computer-internen und Computer-externen Datenflüssen erfolgt (Interface für Bandspeicher, IEEE 488 Bus und Benutzerausgänge). Die im Verkehr mit externen Geräten auftretenden Daten werden über einen Ein/Ausgabe-Speicher von 2K Byte Speicherkapazität den Speicheradressen des CBM-Computers zugewiesen.

Bei CBM-Computern mit eingebautem Bandgerät läuft der Datenaustausch mit dem Bandgerät direkt über den Ein/Ausgabe-Speicher; hierdurch bleibt das Ein/Ausgabe-Tor für einen externen Bandspeicher frei, so daß ein zweites Bandgerät angeschlossen werden kann. Weitere Bandgeräte können über das Ein/Ausgabe-Tor des IEEE 488 Bus angeschlossen werden. (Der IEEE 488 Bus stellt eine in den USA genormte, mehradrige Datenverbindung dar, die vom Computer und externen Geräten für die Abwicklung ihres Datenverkehrs gemeinschaftlich benutzt wird).

Bei einer Speicherkapazität von 65K Byte enthält ein CBM-Computer die in Figur 7-1 dargestellten 6 Speicherblöcke (ROM, RAM, E/A-Speicher). Mit 1K Byte ist eine Speicherkapazität für 1024 Zeichen gemeint. Über die Verwendung der Speicherkapazität wird in Tabelle 7-1 für Blöcke von jeweils 4 K Bytes Auskunft gegeben.



Figur 7-1. Systemaufbau

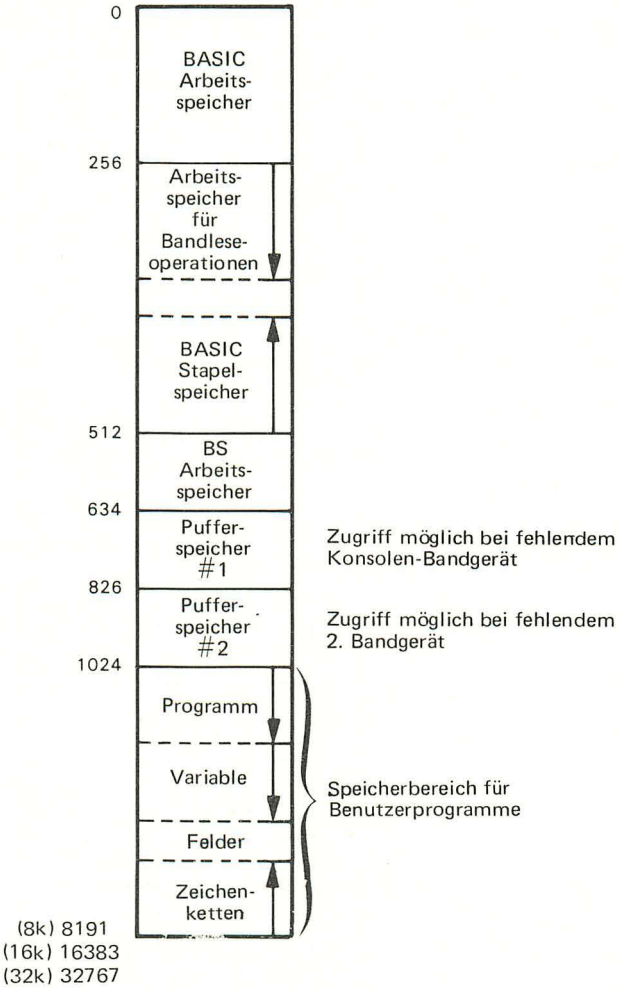
Tabelle 7-1. Speicherbelegung

Speicher-block (je 4K)	Speicher-Typ*	Speicheradresse		Belegung
		Dezimal	Hexadezimal	
0	RAM	0	0000	Arbeitsspeicher
1	RAM	4096	1000	Speicher für Text u. Variable (nur 8K)
2	–	8192	2000	
3	–	12288	3000	
4	–	16384	4000	RAM-Erweiterung
5	–	20480	5000	
6	–	24576	6000	
7	–	28672	7000	
8	RAM	32768	8000	Bildschirmspeicher
9	ROM	36864	9000	
10	ROM	40960	A000	ROM-Erweiterung
11	ROM	45056	B000	BASIC 4.0 (Anfang)
12	ROM	49152	C000	BASIC (haupts. Befehlsinterpret)
13	ROM	53248	D000	BASIC (haupts. math. Progr.)
14	ROM	57344	E000	Bildschirmeditor (2K)
	I/O	59392	E800	Ein/Ausgabespeicher (2K)
15	ROM	61440	F000	Betriebssystem (BS)

*) RAM = Lese/Schreibspeicher, ROM = Lesespeicher, I/O = Ein/Ausgabespeicher

ADRESSEN 0-8191: 8K RAM

Zur Grundausrüstung eines CBM-Computers gehört ein Lese/Schreibspeicher (RAM) mit einer Speicherkapazität von 8K (Adressen 0-8191) oder 16K (Adressen 0-16384) oder 32K (Adressen 0-32767). Die hier beschriebene Konfiguration geht von einem 8K-RAM aus. In allen Fällen ist die Speicherbelegung der ersten 1K Bytes gleich; je größer die Speicherkapazität des RAM, desto größer ist der Speicherplatz für Programme des Benutzers.



Die Speicherplätze 0 bis 255 dienen als Arbeitsspeicher für den BASIC-Übersetzer. Dieser Speicherbereich ist ausführlich in Anhang D dargestellt.

Speicherplätze 256 bis 511 dienen hauptsächlich als BASIC-Stapelspeicher.

Ein Teil des Speicherbereichs von Speicherplatz 256 an aufwärts dient der Bandlese-Routine bei Fehlerkorrekturen und CBM BASIC als erweiterter Pufferspeicher. Der eigentliche Stapelspeicher beginnt mit Speicherstelle 511, d.h. zählt in entgegengesetzter Richtung. Die Speicherbelegung folgt dynamisch den Programmerfordernissen. Ist der Speicherplatz des Stapelspeichers erschöpft, dann erscheint die Fehlermeldung OUT OF MEMORY.

Speicherstellen 512 bis 633 dienen dem Betriebssystem (BS) als Arbeitsspeicher. Die Speicherbelegung dieses Bereichs ist in Anhang D dargestellt.

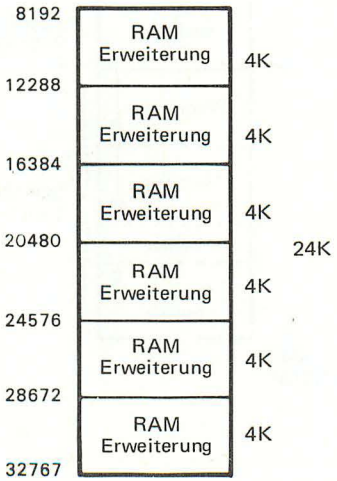
Speicherstellen 634 bis 825 bilden einen 192-Byte-Pufferspeicher für das Konsolen-Bandgerät.

Speicherstellen 826 bis 1023 bilden einen zweiten 192-Byte-Pufferspeicher für ein wahlweise anzuschließendes zweites Bandgerät. Beide Pufferspeicher können zur Speicherung von Assembler-Programmen des Benutzers verwendet werden, wenn kein Konsolen-Bandgerät oder zweites externes Bandgerät vorhanden sind.

Speicherstellen 1024 bis 8191 (oder 16383 bzw. 32767, je nach RAM) stehen zur Speicherung von Programmen des Benutzers zur Verfügung. Die Programmspeicherung beginnt mit Speicherstelle 1024 und geht, wenn nötig, bis zum Speicherende. Nach dem Speicherbereich für Programme folgt der Speicherbereich für Variable und danach der Speicherbereich für Felder (Tabellen). Die Speicherung von Zeichenketten beginnt am Speicherende, d.h. verläuft in entgegengesetzter Richtung. Treffen sich die Zeiger dieser vorwärts- und rückwärtsverlaufenden Speicherbewegungen, dann erfolgt die Fehlermeldung OUT OF MEMORY.

ADRESSEN 8192-32767: 24K RAM ERWEITERUNG

Die Speicherkapazität eines CBM-Computers kann bis auf 32K Bytes erweitert werden. Für die angenommene Grundausstattung mit 8K Bytes ist hierfür ein RAM-Speicher mit 24K Bytes erforderlich; die Speicheradressen laufen dann von 8192 bis 32767:

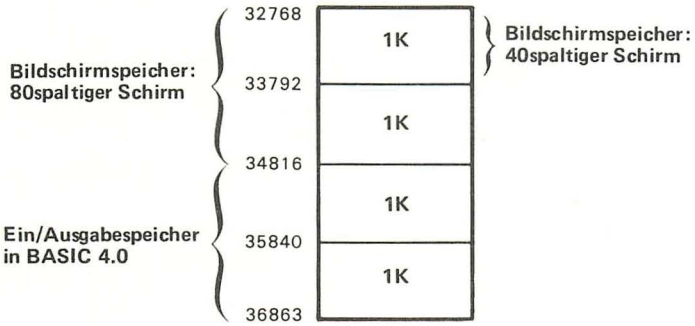


CBM-Computer bieten Speicherkapazitäten, die sich nach folgenden Kombinationen auf Grundausstattung und Erweiterung verteilen können:

RAM Grundausstattung	RAM Erweiterung
8K (0 - 8191)	24K (8192 - 32767)
16K (0 - 16383)	16K (16384 - 32767)
32K (0 - 32767)	

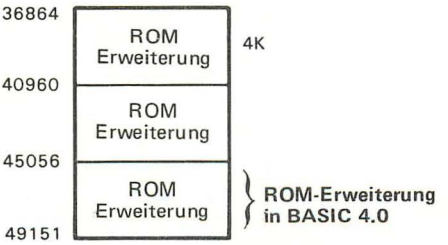
ADRESSEN 32767-36863: 4K RAM

Speicherstellen 32768 bis 33767 bilden den Speicherbereich für 40spaltige, 25zeilige Bildschirme. Der Bildschirmspeicher 80spaltiger Bildschirme umfaßt die Speicherstellen 32768 bis 34767. Der Inhalt jeder dieser Speicherstellen kann mit einer POKE-Anweisung gelesen werden.



ADRESSEN 36864-49151: 12K ROM ERWEITERUNG

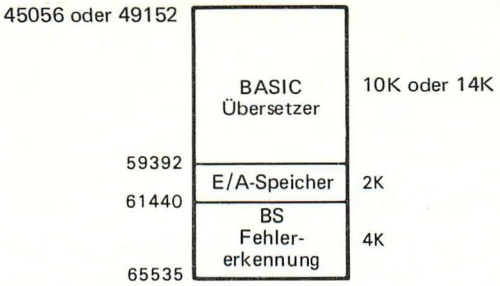
Mit dieser ROM-Erweiterung erhöht sich die Speicherkapazität der ROM-Grundausstattung von 14K auf 26K Bytes.



ADRESSEN 49152-65535: 14K ROM UND 2K E/A-SPEICHER

Die Speicherstellen 49152 bis 59391 und 61440 bis 65535 enthalten den BASIC-Übersetzer und Fehlerdiagnosen des Betriebssystems (BS). Dieser Bereich beginnt in BASIC 4.0 mit Speicherstelle 45056.

Speicherstellen 59392 bis 61439 dienen als Pufferspeicher für den Datenverkehr mit externen Geräten.



Speicherstelle 65535 bildet das Ende des gesamten Speicherbereichs in CBM-Computern.

SPEICHERAUSZÜGE

Ausführliche Speicherauszüge für die verschiedenen Versionen von CBM BASIC finden Sie im Anhang D. Tabelle D-1 enthält den Speicherauszug für Version 2 ROMs. Tabelle D-2 enthält den Speicherauszug für Version 3 ROMs, die in BASIC < 3.0 verwendet werden. Tabelle D-3 zeigt schließlich die letzte Version des Speicherauszugs für BASIC 4.0.

In Tabellen D-1 und D-2 wurden Speicheradressen in dezimaler und hexadezimaler Schreibung angegeben. In PEEK- und POKE-Anweisungen wird die dezimale Speicheradresse verwendet. Gleichfalls in diesen Tabellen finden Sie Beispiele für dezimale und hexadezimale Speicherinhalte.

Abgesehen von Speicherinhalten für Zeiger sind die angegebenen Zahlenbeispiele typisch für ein Leseresultat mit PEEK; alle mit PEEK gelesenen Speicherinhalte liegen im Zahlenbereich 0 bis 255 (0-FF_{16}). Ein Zeiger ist eine 2-Byte-Adresse im Zahlenbereich 0 bis 65535, die im CBM-Computer zweiteilig gespeichert wird: Byte 1 enthält die höherwertigen, Byte 2 die niederwertigen Stellen. Alle 2-Byte-Speicherstellen in den Tabellen des Anhangs D enthalten Speicherinhalte in (Byte 1, Byte 2) -Anordnung. Hier ein Beispiel:

Speicheradresse		Beispiel		Erklärung
Dezimal	Hexadezimal	Dezimal	Hexadezimal	
1-2	0001-0002	826	033A	

Wenn Sie die Speicherstellen 1, 2 mit einer PEEK-Anweisung lesen, erhalten Sie die gespeicherte 16-Byte-Adresse in zwei Teilen; zuerst die niederwertigen Bytes:

```
?PEEK(1)
58
```

und dann die höherwertigen Bytes:

```
?PEEK(2)
3
```

Die beiden Werte 3 und 58 können zunächst in hexadezimaler Form dargestellt werden, um hieraus die dezimale Adresse zu bilden:

höherwertige	niederwertige Stellen		Speicheradresse
$58_{10}=3A_{16}$	$3_{10}=03_{16}$	→	$033A_{16}=826_{10}$

Beachen Sie, daß das Zahlenbeispiel 033A eine 2-Byte-Darstellung der Speicheradresse ist: Byte 1 = 03, Byte 2 = 3A. Stattdessen können Sie auch das höherwertige Byte mit 256 multiplizieren und zum niederwertigen Byte addieren, um die Speicheradresse zu erhalten:

```
?PEEK(1) + 256*PEEK(2)
826
```

Andererseits können Sie eine gegebene 16-Bit-Speicheradresse in die Darstellungsform (Byte 1, Byte 2) bringen und mit der Anweisung POKE abspeichern. Hierzu stellen Sie die dezimale Speicheradresse zunächst hexadezimal dar und wandeln dann Byte 1, Byte 2 der hexadezimalen Darstellung in ihre dezimalen Äquivalente um; hier ein Beispiel:

Speicheradresse		höherwertige		niederwertige Stellen
$59409_{10}=E811_{16}$	→	$E8_{16}=232_{10}$	und	$11_{16}=17_{10}$

Stattdessen können Sie zur Umwandlung den Dezimalwert der Speicheradresse zunächst durch 256 teilen, wobei sie einen Bruchanteil unterdrücken:

		höherwertige Stellen
59409/256=232.06641	→	232

Dann multiplizieren Sie die höherwertigen Stellen (hier 232) mit 256 und ziehen das Produkt vom Dezimalwert der Speicheradresse (hier 59409) ab, um die niederwertigen Stellen zu erhalten:

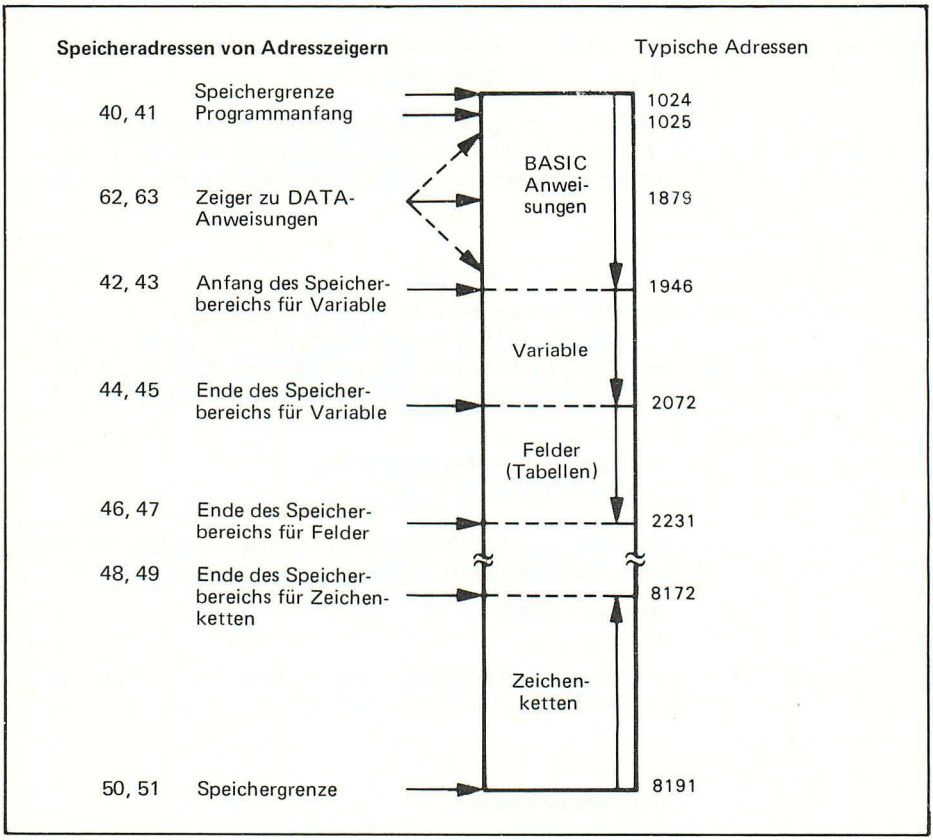
```
232*256=59392
```

		niederwertige Stellen
59409 - 59392	→	17

CBM BASIC-ÜBERSETZER

Der CBM BASIC-Übersetzer führt ein Programm des Benutzers (Quellprogramm) dadurch aus, daß er die Angaben in jeder Programmzeile zunächst dekodiert. Die Programmzeilen des Quellprogramms werden hierbei in eine komprimierte Form gebracht. Bei Eingabe einer Programmzeile über das Tastenfeld haben Sie solange Möglichkeit für Veränderungen, bis die Taste RETURN gedrückt wurde. Nach Eingabe von RETURN sucht der BASIC-Übersetzer im Speicher des Computers nach einer Programmzeile mit der gleichen Zeilennummer. Findet er die gleiche Zeilennummer, dann wird die gespeicherte Programmzeile durch die neue ersetzt. Findet er keine Programmzeile mit gleicher Zeilennummer, dann fügt der BASIC-Übersetzer die neue

Programmzeile derart in die vorhandenen Programmzeilen ein, daß sich eine aufsteigende Ordnung der Zeilennummern ergibt.



Figur 7-2. Wichtigste Adresszeiger im Speicherbereich für Benutzerprogramme (8K ROM)

Die Speicherung von Programmzeilen beginnt am Anfang des Speicherbereichs für Benutzerprogramme, d.h. mit der Speicherstelle 1024. Hierbei übernimmt der BASIC-Übersetzer die Aufgabe, die vier Speicherbereiche für Programmzeilen, Variable, Felder (Tabellen) und Zeichenketten anzulegen und durch Verwaltung von Zeigern zu den einzelnen Speicherstellen Einfügungen und Löschungen zu ermöglichen. Zeiger zu den Trennlinien dieser vier Speicherbereiche werden in 8 Doppel-Speicherstellen gespeichert, wie Figur 7-2 zeigt. Die gleichen Angaben finden Sie auch in den Tabellen des Anhangs D.

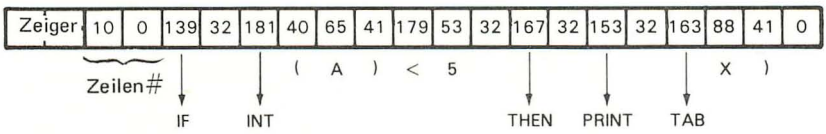
Wir besprechen jetzt, in welchem Format Anweisungen, Variable, Felder (Tabellen) und Zeichenketten in ihren Speicherbereichen gespeichert werden.

Figur 7-3 zeigt die Form, in der BASIC-Anweisungen gespeichert werden. Der Speicherinhalt der Speicherstelle 1024 ist immer Null. Die folgenden beiden Bytes enthalten einen Zeiger auf den Anfang der ersten BASIC-Anweisung. Der Zeiger wird, wie alle anderen Adressen, in 2-Byte-Form dargestellt. Zeiger enthalten als Information die Speicheradresse des nachfolgenden Zeigers. Hierdurch werden Zeiger verkettet. Eine Verkettungsadresse 0 gibt das Ende eines Programmtextes an; d.h. es folgen keine weiteren Verkettungen und keine weiteren Anweisungen. BASIC-Anweisungen werden in aufsteigender Ordnung der Zeilennummern gespeichert, obwohl es Verkettungen zu den nächsten Anweisungen gibt.

Nach der Verkettungsadresse folgt die Zeilennummer der Anweisung, die ebenfalls in (Byte 1, Byte 2)-Form gespeichert ist. Zeilennummern laufen vom Wert 1 (Byte 1 = 0, Byte 2 = 1) bis zum Wert 63999 (Byte 1 = 249, Byte 2 = 255).

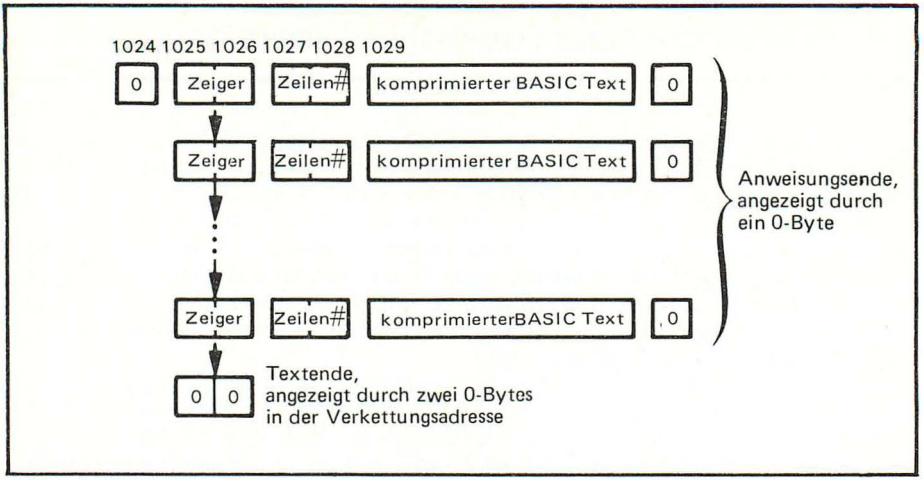
Nach der Zeilennummer folgt der BASIC-Text mit den Programmanweisungen. Wortelemente dieses Textes sind die Reservierten Worte in Tabelle 4-4 und die Abkürzungen für Operatoren nach Tabelle 4-2. Reservierte Worte und logische Operatoren werden in einem komprimierten Format gespeichert: Ihre Darstellung wird in 1 Byte verschlüsselt, dessen höchstwertiges Bit den Wert 1 hat. Die anderen Elemente der BASIC-Sprache werden in Form ihres ASCII-Kode gespeichert; zu diesen Elementen gehören Konstanten, Variable, Felder und Sonderzeichen ohne Operator-Bedeutung. Diese Elemente werden in der Reihenfolge ihres Auftretens im BASIC-Text verschlüsselt. Der kombinierte BASIC-Text besteht schließlich aus den Zahlenkodes 0 bis 255, die nach Tabelle A-1, Anhang A interpretiert werden. Lediglich die Zeichen innerhalb einer Zeichenkette, d.h. zwischen Anführungszeichen, werden im ASCII-Kode verschlüsselt, dessen Interpretation in Tabelle A-4, Anhang A gezeigt ist.

Beachten Sie folgende Besonderheit: Das linke Klammerzeichen ist in der Verschlüsselung der Funktionen TAB und SPC enthalten, während alle übrigen Funktionen ein eigenes Byte für dieses Symbol vorsehen. Sehen Sie folgendes Beispiel:



Zeichen, Operatoren und Reservierte Worte, mit denen BASIC-Anweisungen formuliert werden, heißen auch Schlüsselworte. Ein Vergleich der Tabellen A-1 und A-4 läßt teilweise übereinstimmende Zahlenkodes für Schlüsselworte und Zeichen einer Zeichenkette erkennen. Die Verschlüsselung nach ASCII (Tabelle A-4) bleibt auf Zeichen einer Zeichenkette beschränkt. Im Computer sind 1-Byte-Verschlüsselungen von Schlüsselworten an der 1 im höchstwertigen Bit erkennbar.

Leerstellen in den Programmzeilen des Quellprogramms werden gespeichert, jedoch nicht die Leerstelle zwischen der Zeilennummer und dem Beginn des BASIC-Textes; diese Leerstelle wird bei der Programmauflistung (LIST) wieder hinzugefügt. Durch Vermeidung von Leerstellen können Sie den Speicherbedarf für Ihr Programm verringern, jedoch ist es dann schwieriger, das Programm zu lesen. Speicherplatz können Sie auch dadurch sparen, daß Sie mehr als eine Anweisung in eine Programmzeile aufnehmen, da dann die fünf Bytes für Zeiger, Zeilennummer und das End-Byte nur einmal erforderlich sind:



Figur 7-3. Speicherung von BASIC-Anweisungen (Beispiel)

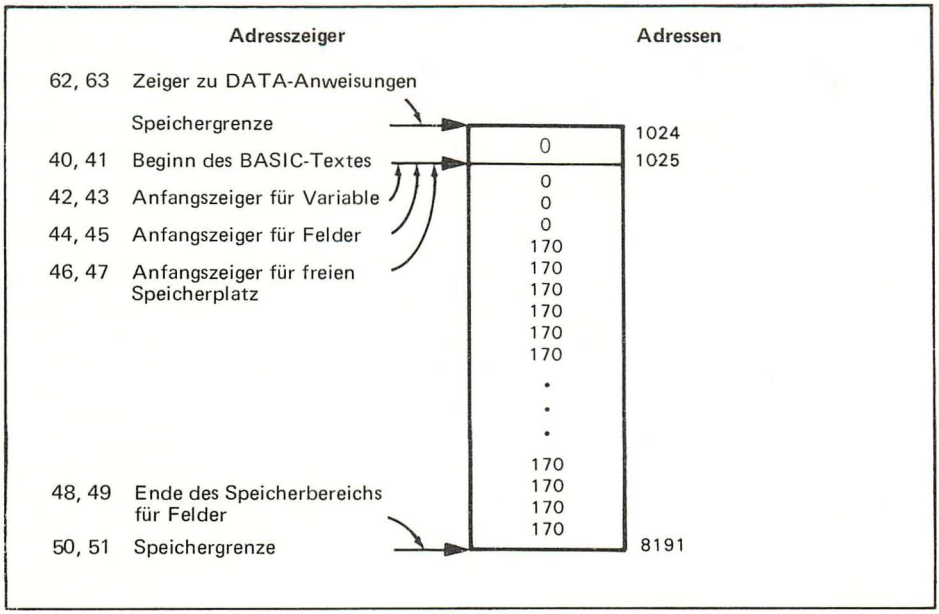
Die Länge jeder Programmanweisung ist unbestimmt; das Ende einer Anweisung wird im Computer mit einem Byte 000 . . . 0 angezeigt. (Der Wert 0 wird im BASIC-Text als Zahlenkode 48 dargestellt). Beim Abarbeiten eines Programms geht der BASIC-Übersetzer von links nach rechts durch den komprimierten BASIC-Text, um die angegebenen Operationen auszuführen. Ein 0-Byte zeigt dem Übersetzer das Ende einer Anweisung; die nächsten vier Bytes enthalten den Zeiger und die Zeilennummer der nächsten Anweisung. Im Gegensatz zum Abarbeiten eines BASIC-Textes bis zum 0-Byte und Übergang zur nächsten Anweisung werden Zeiger benutzt, um Anweisungen aufgrund ihrer bekannten Zeilennummer zu suchen. Drei aufeinanderfolgende 0-Bytes (die beiden Zeiger-Bytes und das Ende-Byte) kennzeichnen das Ende eines BASIC-Textes bei Programmausführung.

Eine Programmspeicherung auf Bandkassette verwendet das gleiche Format, das zur Speicherung im Computer verwendet wurde (Figur 7-3), d.h. auf dem Band ist praktisch ein unveränderter Speicherauszug gespeichert.

Für die Verschlüsselung von BASIC-Anweisungen durch einen Übersetzer gibt es noch keine Normierung. Daher sind Programme von CBM-Computern und Computern anderer Fabrikate, auch wenn sie übereinstimmend in BASIC abgefaßt wurden, nicht austauschbar.

SPEICHERORGANISATION BEI INBETRIEBNAHME

Bei Inbetriebnahme werden in den Speicherbereich für Benutzerprogramme "+"-Symbole (Kode 170) gespeichert, mit Ausnahme der Speicherstellen 1024 bis 1026. Die Speicherstellen 1024 bis 1026 werden auf "0" gesetzt. Die Anfangswerte der Adresszeiger im Speicherbereich für Benutzerprogramme gehen aus Figur 7-4 hervor.



Figur 7-4. Speicherbereich für Benutzerprogramme bei Inbetriebnahme (8K ROM)

Im Verlauf der Programmeingabe ändern sich die Speicherinhalte im Speicherbereich für Benutzerprogramme. Die Änderung der Speicherinhalte geht jedoch nur soweit, wie für die Programmspeicherung erforderlich ist. Die Initiierung der Speicherinhalte mit den Symbolen "+" und "0" wird beim weiteren Arbeiten mit dem Computer nicht wiederholt. Die Information über die Ausdehnung eines Programms im Speicherbereich für Benutzerprogramme ist lediglich in den Adreßzeigern enthalten. Die Löschung von Programmen mit der Anweisung NEW bewirkt lediglich, daß die Adreßzeiger auf ihre Anfangswerte zurückgesetzt werden (s. Figur 7-4). Eine ähnliche Wirkung hat die Anweisung CLR: Statt die Adresszeiger auf Anfangswerte, wie in Figur 7-4 zu bringen, setzt sie lediglich die Adresszeiger für Variable, Felder (Tabellen) und Zeichenketten zurück, als ob die zugehörigen Speicherbereiche leer seien. Die Speicherinhalte werden also nicht gelöscht. Daher können Sie ein Programm oder Variable, die Sie versehentlich mit NEW oder CLR gelöscht haben, durch Lesen des Programms im Speicherbereich für Benutzerprogramme und Wiederherstellung der Adresszeiger vor einem Verlust retten.

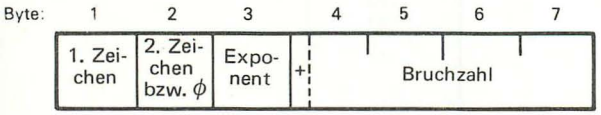
DATENFORMATE

VARIABLE

Variable werden in einem eigenen Bereich innerhalb des Speicherbereichs für Benutzerprogramme gespeichert (Figur 7-2). Variable können vom Typ Ganzzahl,

Gleitkommazahl oder Zeichenkette sein und auch gemischt in ihrem Speicherbereich auftreten. Unabhängig von ihrem Typ belegt jede Variable 7 Bytes im Speicher. Die ersten 2 Bytes enthalten den Namen der Variablen und die weiteren 5 Bytes Angaben über den Inhalt der Variablen. Variable werden in der Reihenfolge ihres Auftretens im Programmablauf in den Variablen Speicher eingetragen. Variable, die nicht im Speicher für Variable stehen, werden behandelt, als ob ihr Inhalt den Wert 0 hat (für numerische Variable) oder eine leere Zeichenkette darstellt (für Zeichenketten-Variable).

VARIABLE VOM TYP GLEITPUNKTZAHL



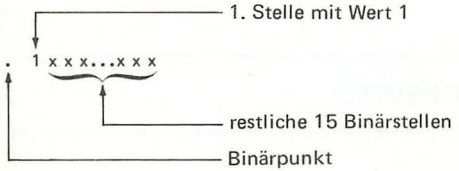
Byte 1 enthält das erste Zeichen des Variablennamens. Byte 2 enthält das zweite Zeichen des Variablennamens, oder falls das zweite Zeichen fehlt, eine 0. Die Zeichen werden im ASCII-Kode gespeichert (Tabelle A-4, Anhang A). Ein Beispiel: Die Variable A wird als 65,00 gespeichert, die Variable A0 als 65,48. Variable für Gleitpunktzahlen werden mit ASCII-Kode-Werten von 90 und kleiner dargestellt.

Bytes 3 bis 7 enthalten den Wert der Variablen in Form einer normalisierten Expotentialdarstellung der Gleitpunktzahl. Zweck der Normalisierung ist, die Darstellung negativer Exponenten zu vermeiden. Hierbei wird jeder gegebene Exponent durch Addition der Zahl 128 in den Bereich positiver Zahlen verschoben:

Exponent: tatsächlicher	gespeicherter	Ungefährer Dezimalwert	
127	255	10^{38}	← maximaler Exponent
34	162	10^{10}	
–1	127	10^{-1}	
–126	2	10^{-38}	
–128	0	10^{-39}	← minimaler Exponent

Byte 3 enthält den Exponenten in normalisierter Form; sein Maximalwert 255 ergibt sich, wenn alle Bits in Byte 3 den Wert 1 annehmen; sein Minimalwert ist 0, bei dem alle Bits in Byte 3 den Wert 0 haben.

Bytes 4 bis 7 enthalten die zählenden Stellen der Mantisse. Die Darstellung der Mantisse wird derart normalisiert, daß der Binärpunkt unmittelbar links der ersten, nicht verschwindenden Binärstelle steht, d.h. die Mantisse wird als Bruchzahl in folgender Form dargestellt:



Der "Binärpunkt" und die "1. Stelle mit Wert 1" werden nicht gespeichert; anstelle des letzteren wird das Vorzeichen gespeichert, und zwar eine 0 für positive, eine 1 für negative Zahlen. Zur Normalisierung einer Mantisse wird der Binärpunkt

nach links bewegt und gleichzeitig der Exponent verkleinert (wodurch sich die Zahl verkleinert) oder der Binärpunkt nach rechts bewegt und der Exponent vergrößert (wodurch sich größere Zahlen ergeben), bis die Mantisse die oben angegebene Form angenommen hat. Der Zahlenwert 0 wird im allgemeinen dadurch dargestellt, daß alle Bits in Bytes 3 bis 7 den Wert 0 annehmen; die oben angegebene Bruchzahl-Darstellung kann jedoch Rundungsfehler enthalten, so daß die sicherste Form der Darstellung des Wertes 0 ein Exponent vom Wert 0 ist (d.h. ein tatsächlicher Exponent mit Wert minus 128).

Es folgen einige Beispiele für die Darstellung von Gleitpunktzahlen, wie sie im Speicherbereich für Variable gespeichert werden:

Zahl	Byte → 3 Exponent	4 ±MSB	5	6 Bruchzahl	7 LSB
1E+38	255	22	118	153	83
1E+10	162	21	2	249	0
1000	138	122	0	0	0
1	129	0	0	00	
0.01	122	35	215	10	62
1E-4	115	81	183	23	90
1E	62	60	229	8	101
1E-39	0	32	0	0	0
0	0	0	0	0	0
-1	129	128	0	0	0
-1000	138	250	0	0	0
-1E+10	162	149	2	249	0
-1E+38	255	150	118	153	83

*) ± MSB = höchstwertige Bits einschließlich Vorzeichen
LSB = niedrigstwertige Bits in der Bruchzahldarstellung

Das folgende kurze Programm erlaubt Ihnen, die Darstellung von Gleitpunktzahlen im Computer zu untersuchen. Zeile 10 fragt Sie nach der Eingabe einer beliebigen Zahl über das Tastenfeld, die Sie mit der Taste RETURN abschließen. Zeile 20 zeigt auf den Anfang des Speicherbereichs für Variable, erhöht um den Wert 2, um die beiden Bytes für den Variablennamen zu überspringen. Zeile 30 druckt die eingegebene Zahl und anschließend ihre mit PEEK aus dem Speicher gelesene Darstellungsform im Computer aus. Zum Beenden des Programms geben Sie lediglich die Taste RETURN ein:

```
10 INPUT A
20 X=PEEK(43)*256+PEEK(42)+2
30 PRINT A;"="PEEK(X);PEEK(X+1);PEEK(X+2);PEEK(X+3)PEEK(X+4)
40 GOTO 10
```

VARIABLE VOM TYP GANZZAHL

Byte	1	2	3	4	5	6	7
	1. Zeichen + 128	2. Zeichen + 128 od. 128	höherwertig. Stellen	niederwertig. Stellen	0	0	0

Byte 1 enthält das erste Zeichen des Variablennamens, dessen ASCII-Kode um den Wert 128 erhöht wurde. Byte 2 enthält das zweite Zeichen des Variablennamens, ebenfalls um den Wert 128 erhöht, oder, wenn das zweite Zeichen fehlt, den Wert 128. Der Name einer Variablen für Ganzzahlen besteht aus Zeichen, deren ASCII-Kode-Werte 176 oder größer sind. Das %-Zeichen im Variablennamen wird unterdrückt. Byte 3 und 4 enthalten die Ganzzahl in (Byte 1, Byte 2)-Darstellung: Die

höherwertigen Stellen befinden sich in Byte 3, die niederwertigen Stellen in Byte 4. Negative Ganzzahlen werden im 2er-Komplement dargestellt; Bit 7 in Byte 3 stellt daher das Vorzeichen dar: eine 0 für positive, eine 1 für negative Zahlen. Bytes 5 bis 7 werden nicht benutzt und ihr Inhalt auf 0 gesetzt.

Im folgenden einige Beispiele für die Darstellung von Ganzzahlen im Speicherbereich für Variable. Für die Inspektion dieses Speicherbereichs können Sie das obengenannte Programm benutzen, wenn Sie die Variable A in A% ändern (Zeilen 10 und 30):

Zahl	Byte → 3	4
32767	127	255 (256•127+255=32767)
32766	127	254
14000	54	176
256	1	0
255	0	255
1	0	1
−1	255	255 (FFF_{16})+1=1
−2	255	254
−32766	128	2
−32767	1281	1

VARIABLE VOM FORMAT ZEICHENKETTEN

Byte	1	2	3	4	5	6	7
	1. Zeichen	2. Zeichen + 128 od. 128	Anzahl der Zeichen	höherw. Stellen	niederw. Stellen	0	0

Byte 1 enthält das erste Zeichen des Variablennamens. Byte 2 enthält das zweite Zeichen des Variablennamens, erhöht um den Zahlenwert 128, oder, bei fehlendem zweiten Zeichen, den Zahlenwert 128. Diese Kombination von Werten des ASCII-Kodes bezeichnet eine Variable für Zeichenketten. Das $-Zeichen im Variablennamen wird unterdrückt. Byte 3 enthält eine Angabe über die Anzahl der Zeichen in der Zeichenkette (Werte 1 bis 255). Dieser Zahlenwert wird von der Funktion LEN gelesen. Bytes 4 und 5 enthalten einen Zeiger zu derjenigen Speicherstelle, an der die eigentliche Zeichenkette gespeichert ist. Der Zeiger wird in der bekannten 2-Byte-Form dargestellt. Bytes 6 und 7 werden nicht benötigt und auf 0 gesetzt.

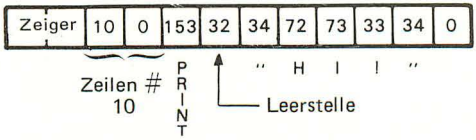
Der Speicherbedarf für Zeichenketten wird dadurch minimiert, daß gleiche Zeichenketten erkannt und nicht mehrmals gespeichert werden.

KONSTANTEN

Konstanten werden in BASIC-Anweisungen gespeichert. Die Verschlüsselung erfolgt nach Tabelle A-1, Anhang A für "Schlüsselworte in CBM BASIC". Hier ein Beispiel:

```
10 PRINT "HI!"
```

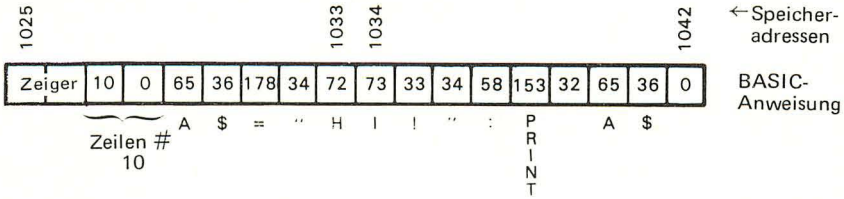
Diese Programmzeile wird im Speicherbereich für Benutzerprogramme in folgender Form gespeichert:



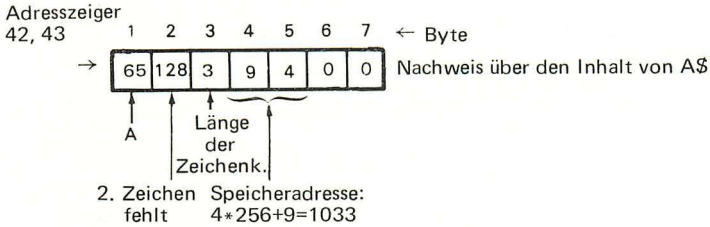
Dagegen wird die folgende Anweisung:

```
10 A$="HI!":PRINT A$
```

in zwei Speicherbereichen abgespeichert. Die eigentliche BASIC-Anweisung wird im Speicherbereich für BASIC-Programme abgespeichert:



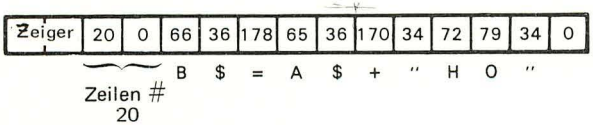
Die im Beispiel verwendeten Speicheradressen nehmen an, daß die angegebene BASIC-Anweisung die erste Eintragung im Speicherbereich für Programme darstellt (beginnend mit Speicherstelle 1025). Außerdem erscheint im Speicherbereich für Variable die folgende Eintragung:



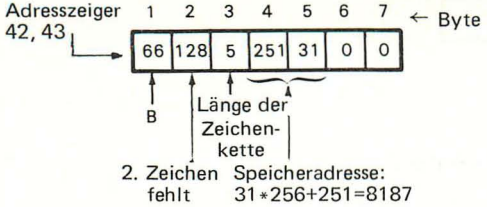
Die Eintragung in den Speicherbereich für Variable enthält den Nachweis darüber, wo die Zeichenkette "HI!" im Computer gespeichert ist, nämlich beginnend mit Speicheradresse 1033 im Speicherbereich für BASIC-Programme. Wenn Sie jedoch eine neue Zeichenkette bilden, wie in folgendem Beispiel:

```
20 B$=A$+"HO"
```

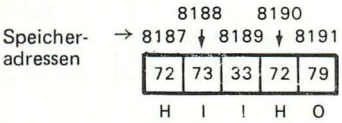
dann hat die Eintragung im Speicherbereich für BASIC-Programme folgende Form:



und die Eintragung im Speicherbereich für Variable die folgende Form:



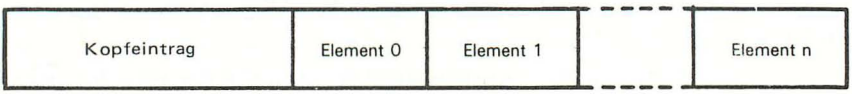
Diesmal weist der Zeiger auf eine Speicherstelle im Speicherbereich für Zeichenketten, wo folgende Zeichenketten abgespeichert sind:



Die im Beispiel verwendete Anfangsadresse 8187 nimmt einen CBM-Computer mit 8K ROM-Speicher an. In diesem Fall hat die größtmögliche Speicherstelle die Adresse 8191.

SPEICHERUNG VON TABELLEN (FELDERN)

Tabellen werden im Speicherbereich für Felder (Figur 7-2) gespeichert. Tabellen können als Eintragungen Gleitpunktzahlen, Ganzzahlen oder Zeichenketten enthalten und werden in der Reihenfolge gespeichert, in der sie im Programm auftreten. Der Typ der Eintragungen wird durch den Typ des Variablennamens gekennzeichnet, mit dem die Tabelle bezeichnet wurde. Namen für Tabellen und Namen für Variable werden in gleicher Form verschlüsselt. Die Speicherung einer Tabelle besteht aus einem Kopfeintrag, gefolgt von den einzelnen Eintragungen (oder Elementen) der Tabelle:



Bei Zeichenketten werden die Elemente in umgekehrter Reihenfolge abgespeichert.

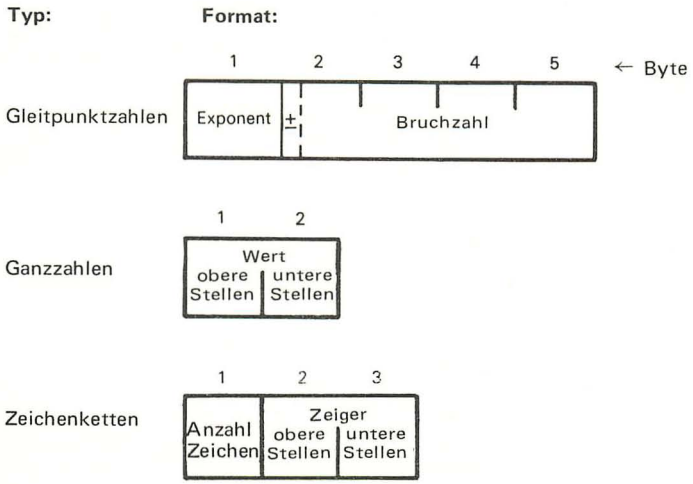
KOPFEINTRAG

Alle Tabellen benutzen das gleiche Format für den Kopfeintrag. Der Kopfeintrag besteht aus 7 Bytes mit zusätzlich 2 Bytes für jede Dimension, die den Wert 1 überschreitet.

Tabellenelemente vom Typ Gleitpunktzahl werden mit Variablen für Gleitpunktzahlen dargestellt; daher werden von jeder Gleitpunktzahl 5 Bytes belegt. Tabellenelemente vom Typ Ganzzahl belegen 4 Bytes, während Tabellenelemente vom Typ Zeichenkette 5 Bytes benötigen; in allen Fällen werden 0-Bytes eliminiert.

Im Kopfeintrag enthalten Bytes 1 und 2 den Namen der Tabelle (des Feldes) Bytes 3 und 4 enthalten die Anzahl der Speicherstellen, die von der Tabelle belegt werden. Ein Beispiel: A (0) würde 12 Bytes belegen: 7 Bytes für den Kopfeintrag und 5 Bytes für jedes Tabellenelement. Die Anzahl der von der Tabelle belegten Bytes wird in 2-Byte-Form gespeichert (Bytes 3 und 4). Byte 5 enthält die Anzahl der Dimensionen der Tabelle. Ein Beispiel: A (5) hat eine Dimension, d.h. Byte 5=1, A (10, 10, 2) hat 3 Dimensionen, d.h. Byte 5=3. Für 1-dimensionale Tabellen geben Bytes 6 und 7 die Anzahl der Tabellenelemente an; das ist die in einer DIM-Anweisung in Klammern angegebene Zahl + 1. Ein Beispiel: Für DIM A (60) enthalten Bytes 6, 7 den Wert 61. Wird die Tabelle nicht in einer DIM-Anweisung angekündigt, dann wird automatisch eine Tabelle mit 11 Elementen angenommen. Die Anzahl der in der DIM Anweisung angegebnen Dimensionen wird in Byte 5 gespeichert. Die Anzahl der Elemente in jeder Dimension wird in Byte 6 und 7 gespeichert. Für jede weitere Dimension werden jeweils 2 Bytes zur Angabe der Anzahl der Elemente bereitgestellt, wobei jeweils die letzte Dimensionsangabe in Bytes 6 und 7 steht. Ein Beispiel: Bei DIM A (1, 2, 3) steht in Bytes 6,7 : 3 + 1 = 4, in Bytes 8,9: 2 + 1 = 3 und in Bytes 10,11: 1 + 1 = 2.

Das Format von Tabellenelementen wird für jeden Typ noch einmal dargestellt:



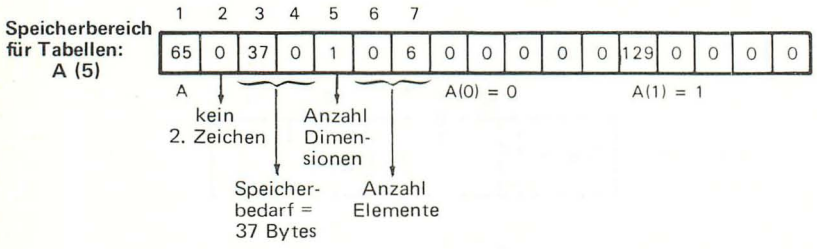
Die Byte-Länge des Kopfeintrags errechnet sich zu 5 Bytes + zweimal der Anzahl der Dimensionen der Tabelle. Der Speicherbedarf der Tabellenelemente kann

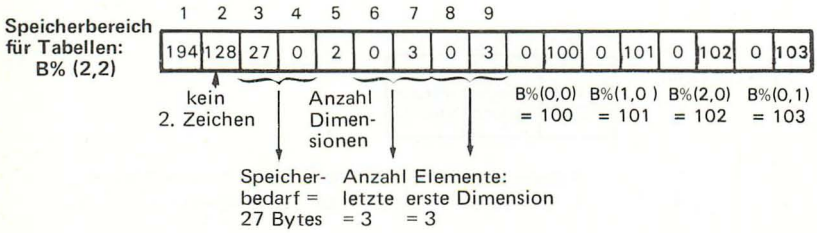
aus der Anzahl der Bytes je Element (5 für Gleitpunktzahlen, 2 für Ganzzahlen, 3 für Zeichenketten) multipliziert mit der Anzahl der Elemente, berechnet werden. Der gesamte Speicherbedarf einer Tabelle in Bytes für Kopfeintrag und Tabellenelemente wird in Byte 4 gespeichert.

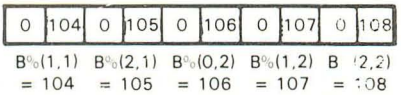
Mit dem folgenden Programm können Eintragungen in den Speicherbereich für Tabellen (Felder) untersucht werden:

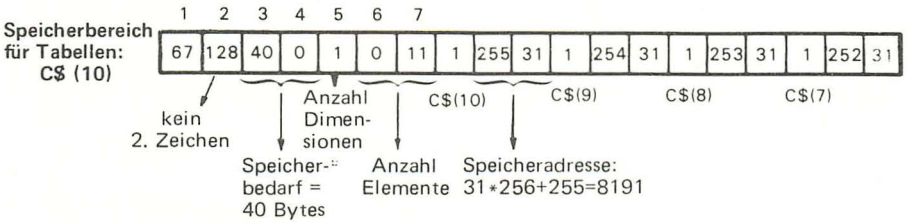
```
10 DIM A(5),B%(2,2),C$(10)  :REM ANLEGEN VON VERSUCHS-TABELLEN
20 FOR I=0 TO 5 : A(I)=I : NEXT
30 FOR I=0 TO 2 : FOR J=0 TO 2 : B%(J,I)=100+3*I+J :NEXT I,J
40 FOR I=0 TO 10 :C$(I)=CHR$(ASC("A")+I) : NEXT I
45 REM ADRESSZEIGER ZUM ENDE DES SPEICHERBEREICHS FUER VARIABLE LESEN:
50 X=PEEK(45)*256+PEEK(44)
55 REM ADRESSZEIGER ZUM ENDE DES SPEICHERBEREICHS FUER TABELLEN LESEN:
60 Y=PEEK(47)*256+PEEK(46)
70 FOR=X TO Y
80 PRINT I , PEEK(I)
90 GET D$ : IF D$="" GOTO 90
100 NEXT I
```

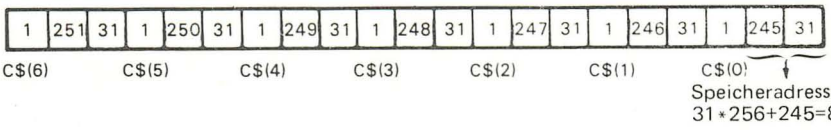
Das Programm verwendet drei Typen von Tabellen, die in Zeile 10 mit DIM dimensioniert werden. Zeile 20 füllt Tabelle A mit den Zahlen 0 bis 5. Zeile 30 füllt die Tabelle C$ mit den Zeichenketten A bis K. Zeilen 50 und 60 lesen die Adresszeiger zum Ende des Speicherbereichs für Variable und zum Ende des Speicherbereichs für Tabellen. Die Bildschirmdarstellung hält bei jeder Speicheradresse an; zur Darstellung der nächsten Speicheradresse drücken Sie irgendeine Taste. Benutzen Sie die folgenden Darstellungen des Speicherbedarfs der Tabellen, um den Adreßzeiger zum Anfang des Speicherbereichs für Tabellen zu finden, der zugleich der Endzeiger des Speicherbereichs für Variable ist. Die Speicherstellen zeigen folgende Belegung:

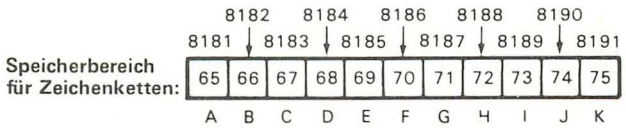






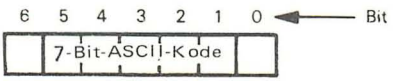






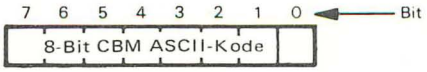
ZEICHENKODIERUNGEN IN CBM-COMPUTERN

Für die Verschlüsselung von Zeichen im digitalen Informationsaustausch zwischen Geräten wurde in den USA der 7-Bit-ASCII-Kode normiert, mit dem 128 Zeichen dargestellt werden können. Die Bedeutung von ASCII ist American Standard Code for Information Interchange. Tabelle A-2 in Anhang A enthält die Zusammenstellung der in ASCII verschlüsselbaren Zeichen. Die Zählung der Bits beginnt bei 0 (niederwertigstes Bit) und geht bis 6 (höchstwertiges Bit):

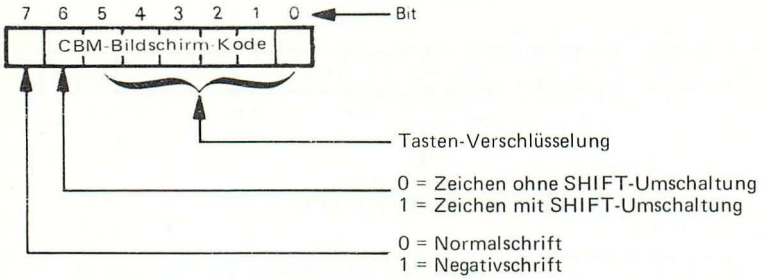


Die ersten 32 Kodes sind für nicht-druckbare Kontrollzeichen vorgesehen, mit denen Formatanweisungen für Darstellungen von Informationen gegeben werden.

Zur Speicherung von Zeichen in CBM-Computern wird eine 8-Bit-Version des ASCII-Kodes verwendet, mit der bis zu 256 Zeichen verschlüsselt werden können. Tabelle A-1 in Anhang A enthält eine Zusammenstellung der im 8-Bit-ASCII-Kode verschlüsselten Zeichen und Schlüsselworte. Diese Kodierung wird insbesondere zur komprimierten Darstellung von BASIC-Anweisungen im Speicher des Computers verwendet; zur Kennzeichnung verschlüsselter BASIC-Anweisungen wird Bit 7 auf den Wert 1 gesetzt. Außerhalb dieses Anwendungsfalls wird der 8-Bit-Kode im Arbeitsspeicher nach Tabelle A-4 interpretiert.



Für die Verschlüsselung von Zeichen im Bildschirmspeicher wird ein weiterer 7-Bit-Kode verwendet, der in Tabelle A-3, Anhang A dargestellt ist. Dieser Kode ist in den Speicheradressen 32768 bis 33767 des Bildschirmspeichers anzutreffen:



Der 8-Bit-CBM-Bildschirm-Kode entsteht aus dem 8-Bit-CMB-ASCII-Kode, wenn Bit 7 (d.h. das achte Bit) im CBM-ASCII-Kode nach rechts in die Stelle von Bit 6 verschoben wird, wobei ein in Bit 6 vorhandener Wert überschrieben und in Bit 7 eine 0 eingetragen wird. Das folgende Beispiel zeigt, wie aus dem linksstehenden Kode durch die genannte Verschiebung der rechtsstehende Kode hervorgeht:

Zeichen	Darstellung im	
	CBM ASCII-Kode	CBM-Bildschirm-Kode
A	01000001	00000001
SHIFT + A (♠)	11000001	01000001
1	00110001	00110001
SHIFT + 1 (▔)	10110001	01110001

Bei Bildschirmausgaben mit der Anweisung PRINT führt der CBM-Computer automatisch die Zeichenumwandlung in dem Bildschirm-Kode durch. Lediglich bei Anwendung der Anweisungen PEEK und POKE auf die Speicherstellen 32768 bis 33767 im Bildschirm-Speicher müssen Unterschiede in den Zeichensätzen beachtet werden. Als Antwort auf die Anweisung POKE 59468,14 setzt der Bildschirm-Kode ein Bit, um den alternativen Zeichensatz zu kennzeichnen. Der alternative Zeichensatz ist auch in Tabelle A-4 aufgeführt.

PROGRAMMIERUNG IM ASSEMBLER

In CBM BASIC können kleine, in Assembler-Sprache des Mikroprozessors 6502 geschriebene Programme ablaufen. Assembler-Programme laufen schneller ab und benötigen weniger Speicherplatz als die gleichen, in BASIC-ausgedrückten Programme. In folgenden Situationen kann es für Sie wünschenswert sein, ein Programm in Assembler-Sprache auf dem CBM-Computer zu verwenden:

1. Die gewünschte Operation ist in BASIC nicht schnell genug.
2. Die gewünschte Operation kann nicht in CBM BASIC dargestellt werden.
3. Die gewünschte Operation belegt zuviel Speicherplatz im BASIC-Programm.

4. Die Assembler-Sprache ermöglicht eine direktere Formulierung der Aufgabe als BASIC. Einige Ein/Ausgabe-Operationen fallen in diese Kategorie.

Ein Programm in Assembler-Sprache wird über POKE-Anweisungen in den Speicher des Computers geladen. Für die Aufnahme von Assembler-Programmen ist kein besonderer Speicherbereich vorbehalten. Für die Wahl des Speicherbereichs haben Sie folgende Möglichkeiten:

1. Speicherbereich für Bandgeräte.
   Wenn Sie kein zweites Bandgerät betreiben, können Sie den zugeordneten 192-Byte-Pufferspeicher zur Speicherung eines Assembler-Programms verwenden. Die Adressen dieses Bereichs sind 826 bis 1017. Wenn darüber hinaus das Konsolen-Bandgerät während des Programmablaufs nicht benutzt wird, steht ein weiterer 192-Byte-Pufferspeicher mit den Speicheradressen 636 bis 825 zur Verfügung.

2. Speicherbereich nahe der Speicherobergrenze.
   Die Speicherobergrenze, d.h. die physikalische Grenze der Speicherkapazität eines CBM-Computers, wird durch einen Adreßzeiger in den Speicherstellen 52,53 markiert. In einem CBM-Computer mit 8K ROM zeigt der Adreßzeiger auf 8192. Sie können vorübergehend den Adreßzeiger zur Speicherobergrenze derart verschieben, daß ein Speicherbereich für das Assembler-Programm entsteht. Ein Beispiel: Wenn Sie 1000 Bytes benötigen, verringern Sie im Adreßzeiger die Adresse der Speicherobergrenze von 8192 auf 8192 – 1000 = 7192:

   obere **untere Stelle**

   $7192_{10} = 1C18_{16} \rightarrow 1C_{16} = 28_{10}$ und $18_{16} = 24_{10}$

   In Speicherstelle 52 wird demnach die Zahl 24, in Speicherstelle 53 die Zahl 28 gespeichert. Verwenden Sie hierfür folgende Anweisungen:

```
10 REM UNTERES/OBERES BYTE DES SPEICHERENDE-ZEIGERS LESEN:
20 AU=PEEK(52) : AO=PEEK(53)
25 REM SPEICHERENDE-ZEIGER AUF 7192 SETZEN:
30 POKE 52,24 : POKE 53,28
.
200 REM SPEICHERENDE-ZEIGER ZURUECKSETZEN:
210 POKE 52,AU : POKE 53,AO
220 END
```

3. Speicherbereich für BASIC-Programme.
   Sie können einen Speicherblock mit DATA-Schein-Anweisungen belegen und diese Speicherstellen für das Assembler-Programm verwenden. Im allgemeinen sind zwischen dem Ende des BASIC-Programms und dem Anfang des Speicherbereichs für Variable nur wenige Speicherstellen frei. Sie müssen daher sehr vorsichtig sein, daß Ihr Assembler-Programm und der BASIC-Übersetzer sich nicht überkreuzen.
   Ein Programm in Assembler-Sprache kann mit Hilfe des CBM-BASIC-Übersetzers in einen ausgewählten Speicherbereich geladen werden. Dieser Vorgang ist stark vereinfacht und besteht in Anwendung der Anweisung POKE, mit der die Befehle der Maschinensprache des Mikroprozessors 6502 in Form von Dezimalzahlen an ihre Speicherstellen gebracht werden. Dieses durch Dezimalzahlen verschlüsselte Assembler-Programm können Sie dann über DATA-Anweisungen in Ihr BASIC-Hauptpro-

gramm übernehmen, wo es anschließend mit READ-Anweisungen gelesen und mit POKE-Anweisungen in einer Programmschleife an die vorgesehenen Speicherplätze im Computer gebracht wird.
Zum Übergang von Ihrem BASIC-Hauptprogramm zum Assembler-Programm stehen die beiden Funktionen SYS und USR zur Verfügung. Mit SYS wird die Kontrolle des Programmablaufs an ein Assembler-Programm wie ein Unterprogramm angesprochen, wobei ein Parameter an das Assembler-Programm übergeben und als Rückmeldung ein Parameter vom Assembler-Programm an das BASIC-Hauptprogramm zurückgemeldet werden kann. Das Assembler-Programm muß nach Abschluß seiner Operationen die Programmkontrolle über eine RTS-Anweisung an das BASIC-Hauptprogramm abtreten.

SYS

Format: SYS(adresse)

Zweck: Aufruf eines Unterprogramms in Assembler-Sprache.

Bemerkung: Das Argument "adresse" kann eine Konstante, Variable oder ein Ausdruck sein, mit dem die Anfangsadresse des Assembler-Programms bezeichnet wird. Der Wert von "adresse" muß im Bereich 0 bis 65535 liegen. SYS ist vergleichbar mit GOSUB, jedoch mit dem wichtigen Unterschied, daß der Schutz in CBM BASIC gegen Programmierfehler des Benutzers außer Kraft gesetzt ist. Die Fehlerbehebung in Assembler-Programmen gestaltet sich schwieriger als die Fehlerbehebung in BASIC-Programmen. Mit der Anweisung RTS veranlassen Sie die Rückkehr aus dem Assembler-Programm in das BASIC-Programm. Der Austausch von Parametern zwischen dem BASIC-Programm und dem von SYS aufgerufenen Assembler-Programm erfolgt über die Anweisungen PEEK und POKE.

```
DIREKTER DIALOG                    PROGRAMMFORM

SYS(826)                           55 SYS(826)
                                   55 SYS(A+14)
```

USR

Format: USR(parameter)

Zweck: Aufruf eines Unterprogramms in Assembler-Sprache mit Parameterübergabe.

Bemerkung: Das Argument "parameter" enthält den an das Assembler-Programm zu übergebenden Parameter. Vor Aufruf von USR muß die Anfangsadresse des Assembler-Programms in den Speicherstellen 1 und 2 abgelegt worden sein. Ein Beispiel: Wenn das Unterprogramm in Assembler auf Bandgerät # 2 abgespeichert ist, würden Sie folgende Anweisungen verwenden:

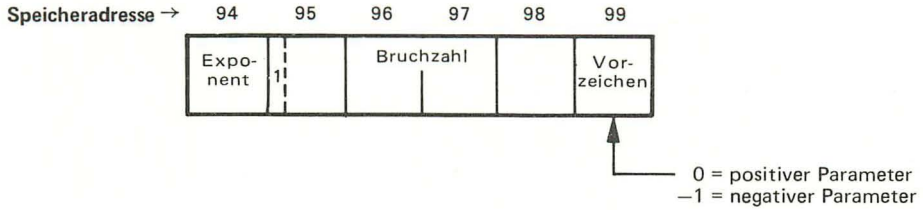
```
10 POKE 1,58
20 POKE 2,3
```

obere **untere Stelle**

$826_{10}=033A_{16}=3A_{16}=58_{10}$ und $03_{16}=3_{10}$

Der von USR an das Assembler-Programm übergebene Parameter wird in einem 6-Byte-Speicherbereich für Gleitpunktzahlen abgelegt, der die Speicheradres-

sen 94 bis 99 ($5E_{16}$–63_{16}) trägt. Parameter werden in diesem Speicherbereich mit folgendem Format abgespeichert:



Wie bei Variablen für Gleitpunktzahlen wird der Exponent normiert, d.h. in den Zahlenbereich von 0 bis 255 verschoben, während die Bruchzahl derart normiert wird, daß das höchstwertige Bit in Byte 95 immer den Wert 1 annimmt. Byte 99 speichert das Vorzeichen der Bruchzahl.

Das mit USR aufgerufene Assembler-Unterprogramm liest den Übergabeparameter aus den eben genannten Speicheradressen, die es auch zur Abspeicherung eines an das BASIC-Hauptprogramm zu übergebenden Parameters verwendet.

DATEIEN FÜR WAHLFREIEN ZUGRIFF

Dateien mit wahlfreiem Zugriff entstehen durch direkte Adressierung von Speicherblöcken auf Disketten und in Pufferspeichern. Jeder Datenblock auf einer Diskette belegt einen einzelnen Sektor. Dateien für wahlfreien Zugriff adressieren direkt Datenblöcke auf der Diskette mit Hilfe ihrer Spur- und Sektor-Adressen. Ebenso können Pufferspeicher von Floppy-Disk-Speichern direkt adressiert werden. Zur Erinnerung: Jedem Floppy-Disk-Speicher sind 16 Pufferspeicher für 256 Byte zugeordnet.

Dateien für wahlfreien Zugriff werden durch eine Reihe von Unterprogrammen gebildet, mit denen der direkte Zugriff auf die Oberfläche der Diskette und auf die Pufferspeicher möglich ist. Diese Unterprogramme sind prinzipiell die gleichen wie beim Arbeiten mit Sequentiellen und Relativen Dateien; jedoch wird durch Ihr Programm die Struktur der Felder/Aufzeichnungen/Dateien bestimmt.

Das Arbeiten mit Dateien für wahlfreien Zugriff erfordert viel Erfahrung. Sie haben mit dem Arbeitswissen derjenigen umzugehen, die die Programmlogik für Sequentielle und Relative Dateien entworfen haben. Derartige Personen sind professionelle System-Programmierer. Ehe Sie nicht vergleichbare Programmiererfahrungen gesammelt haben, werden Sie nicht viel Erfolg mit den in diesem Abschnitt gegebenen Informationen haben.

Wahlfreier Zugriff auf Disketten wird mit PRINT#-Anweisungen programmiert, die geeignet verschlüsselte Zeichenketten in ihrer Parameterliste enthalten. Diese PRINT#-Anweisungen arbeiten mit dem Befehlskanal, d.h. mit der Sekundäradresse 15. Logische Dateien für wahlfreien Zugriff werden mit besonderen Pufferspeichern für Floppy Disk-Geräte geöffnet, die jeder logischen Datei über die Sekundäradresse zugewiesen werden. Die Parameter in einer PRINT#-Anweisung verwenden die Sekundäradresse, um logische Dateien und die ihnen zugeordneten Pufferspeicher zu identifizieren.

Logische Dateien für wahlfreien Zugriff werden in folgender Form mit einer OPEN-Anweisung geöffnet:

100 OPEN ld, gn, sa, "#[pu]"

mit:

ld	logische Dateinummer
gn	Gerätenummer, typischer Wert 8
sa	Sekundäradresse im Bereich 2 bis 14
pu	Pufferspeicher-Nummer im Bereich 3 bis 15. Bei fehlender Angabe wird der nächste verfügbare Pufferspeicher der Sekundäradresse sa zugeordnet.

Unmittelbar nach dem Öffnen einer Datei für wahlfreien Zugriff können Sie mit einer GET#-Anweisung die zugewiesene Nummer des Pufferspeichers lesen. Die GET#-Anweisung muß jedoch vor jeder anderen Ein/Ausgabe-Anweisung ausgeführt werden. Hier ein Programmbeispiel:

```
10 REM  ZUORDNUNG VON PUFFERSPEICHER #5 MITTELS
15 REM  SEKUNDAERADRESSE 4 ZUR LOGISCHEN DATEI 2:
20 OPEN 2,8,4,"#5"
30 PRINT DS$   :REM EIN/AUSGABE-OPERATION PRUEFEN
35 REM ZUR PRUEFUNG PUFFERSPEICHER # DARSTELLEN:
40 GET#2,A$ : PRINT ASC(A$)
50 PRINT DS$   :REM EIN/AUSGABE-OPERATION PRUEFEN
60 CLOSE 2
70 STOP
```

Befehle für den Zugriff auf Dateien mit wahlfreiem Zugriff werden über PRINT#-Anweisungen ausgegeben, wie folgendes Format zeigt:

```
10 OPEN ld, 8, 15
20 PRINT #ld, "parameter"
```

Der Zugriff auf die Datei wird in "parameter" formuliert; diese Angabe besteht aus zwei Teilen: einem Befehl und einer Parameter-Liste. Der Befehl wird entweder ausgeschrieben und muß in diesem Fall mit einem Doppelpunkt enden, oder der Befehl wird in abgekürzter Form angegeben, und dann wird der Anfang der Parameter-Liste beim vierten Zeichen in "parameter" angenommen. Zur Beschreibung der Parameter werden wir folgende Abkürzungen verwenden:

sa	Sekundäradresse
lf	logische Dateinummer
dr	Laufwerk (0 oder 1) im Floppy Disk-Speicher
t	Spurnummer auf der Diskette
s	Sektornummer auf Spur t
p	Zeiger in den Pufferspeicher mit Werten zwischen 0 und 255
adl	unteres Byte einer 2-Byte-Speicheradresse
adh	oberes Byte einer 2-Byte-Speicheradresse
nc	Zeichenzahl (1 bis 34)
data	Zeichenkette mit nc Zeichen

Die Parameter adl, adh und nc müssen als Argumente der Funktion CHR$ angegeben werden. Besitzt beispielsweise adl den Wert 123, dann muß es als CHR$ (123) angegeben werden.

LESEN EINES SEKTORS AUF DISKETTE

Folgende Anweisung liest jeden Sektor auf einer Diskette in den Pufferspeicher:

PRINT# lf, "BLOCK-READ:sa,dr,t,s"
oder: PRINT# lf, "B-Rsa,dr,t,s"

Das folgende Programmbeispiel öffnet die logische Datei 2, weist Pufferspeicher # 5 der Sekundäradresse 4 zu und liest dann Sektor 0 auf Spur 18 der Diskette 1 in den Pufferspeicher # 5:

```
10 REM  ZUORDNUNG VON PUFFERSPEICHER #5 MITTELS
15 REM  SEKUNDAERADRESSE 4 ZUR LOGISCHEN DATEI 2:
20 OPEN 2,8,4,"#5"
30 REM SEKTOR 0 AUF SPUR 18 DER DISKETTE 1 IN PUFFER #5 LESEN:
40 OPEN 15,8,15RINT ASC(A$)
50 PRINT#15,"B-R4,1,18,0"
60 REM PUFFERINHALT DES 256 BYTE PUFFERS IN 8 SPALTEN ZU JE 32
70 REM ZAHLEN AUF DEM BILDSCHIRM DARSTELLEN:
75 PRINT"■";
80 FOR I=1 TO 8
90 FOR J=1 TO 32
100 GET#2,A$ : IF A$="" THEN 100
110 PRINT ASC(A$);
120 NEXT J
130 PRINT
140 NEXT I
150 CLOSE 2
160 CLOSE 15
170 STOP
```

SCHREIBEN IN EINEN SEKTOR AUF DISKETTE

Die folgende Anweisung gestattet, den Inhalt eines Pufferspeichers in einen angegebenen Sektor auf einer Diskette zu schreiben:

PRINT# lf,"BLOCK-WRITE:sa,dr,t,s"
oder: PRINT# lf,"B-Wsa,dr,t,s"

Der folgende Programmauszug öffnet die logische Datei 2, weist den Pufferspeicher # 8 der Sekundäradresse 7 zu und schreibt dann den Inhalt des Pufferspeichers # 8 in Sektor 10 auf Spur 35 der Diskette 0:

```
200 OPEN 2,8,7,"#8"
210 OPEN 15,8,15
220 REM  ANWEISUNGEN ZUM SCHREIBEN IN
225 REM  PUFFER #8 MUESSEN HIER FOLGEN
300 PRINT#15,"B-W4,0,35,0"
310 CLOSE 2
320 CLOSE 15
330 STOP
```

LESEN UND AUSFÜHREN EINES PROGRAMMS: BLOCK EXECUTE

Diese Anweisung hat Ähnlichkeit mit der Anweisung BLOCK READ, jedoch werden die von einem Sektor gelesenen Daten als Objektkode eines Assembler-Programms interpretiert. Sobald das Programm geladen ist, wird es ausgeführt. Das Programm muß mit einer RCS-Anweisung enden. Die Anweisung BLOCK EXECUTE hat folgendes Format:

PRINT# lf,"BLOCK-EXECUTE:sa,dr,t,s"
oder: PRINT# lf,"B-Esa,dr,t,s"

PUFFERZEIGER: BUFFER POINTER

Diese Anweisung bewegt den Zeiger vom Anfang eines Pufferspeichers zu jeder Zeichenposition innerhalb des Pufferbereichs. Die Anweisung hat folgendes Format:

```
      PRINT# lf,"BUFFER-POINTER:sa,p"
oder: PRINT# lf,"B-Psa,p"
```

Als Beispiel für die Anwendung kann folgende Programmzeile in das Beispielprogramm für BLOCK READ eingefügt werden, um den Pufferspeicher auf Zeichen 26 zu setzen:

```
55 PRINT#15,"B-P4,26"
```

BLOCKBELEGUNGSLISTE: BLOCK ALLOCATE

Diese Anweisung bringt die Block-Belegungsliste (BAM) auf den letzten Stand, um die Belegung des augenblicklich benutzten Blocks zu zeigen. Die Block-Belegungsliste wird beim Schließen der logischen Datei auf Diskette geschrieben. Wenn der gewünschte Block bereits belegt ist, dann wird der nächste verfügbare Block festgestellt und die Fehlermeldung NO BLOCK ausgegeben. Sind keine Blöcke verfügbar, dann meldet die Anweisung BLOCK ALLOCATE für die Spur t und den Block s einen Wert 00. Die Anweisung hat folgendes Format:

```
      PRINT# lf,"BLOCK-ALLOCATE:dr,t,s"
oder: PRINT# lf,"B-Adr,t,s"
```

SCHREIBEN IN DEN PUFFERSPEICHER FÜR DISKETTEN

Die Anweisung MEMORY WRITE schreibt Daten in den Pufferspeicher der Diskette. Die Anweisung hat folgendes Format:

```
PRINT# lf,"M-W"adl/adh/nc/data
```

Tabelle 7-2. Startadressen der Pufferspeicher für CBM Floppy Disk-Speicher

Pufferspeicher #	Modelle CBM 3040, CBM 8050	
	Hexadezimal	Dezimal
0	1000	4096
1	1100	4352
2	1200	4608
3	1300	4864
4	2000	8192
5	2100	8448
6	2200	8704
7	2300	8960
8	3000	12288
9	3100	12455
10	3200	12800
11	3300	13056
12	4000	13312
13	4100	13568
14	4200	13824
15	4300	14080

Startadressen von Pufferspeichern in CBM-Computern sind in Tabelle 7-2 zusammengestellt. Die Adressen sind unregelmäßig verteilt.

Nehmen Sie als Beispiel die vier Daten-Bytes 32, 0, 17, 96, die in den Puffer 2 des Floppy Disk-Speichers CBM 3040 geschrieben werden sollen. Aus Tabelle 7-2 entnehmen wir als Startadresse für den Pufferspeicher 1800_{10}. Daher benötigen wir folgende PRINT#-Anweisung:

```
100 PRINT#15,"M-W"CHR$(00)CHR$(18)CHR$(32)CHR$(0)CHR$(17)CHR$(96)
```

LESEN EINES DISKETTEN-PUFFERSPEICHERS

Die Anweisung MEMORY READ zum Lesen von 1 Byte Daten von einer Diskette hat folgendes Format:

```
PRINT# lf,"M-R"adl/adh
```

Die Adresse des zu lesenden Byte wird über CHR$-Funktionen in der Parameterliste mitgeteilt. Das Byte selbst wird dann mit einer GET#-Anweisung über Befehlskanal 15 gelesen. Eine anschließende INPUT#-Anweisung wird nicht richtig ausgeführt, solange nicht eine Anweisung wie MEMORY READ, MEMORY WRITE oder MEMORY EXECUTE zum Zugriff auf die Datei erfolgt ist. Im folgenden Beispiel wird 1 Byte vom Pufferspeicher mit der Adresse 1808 gelesen:

```
100 PRINT#15,"M-R"CHR$(8)CHR$(18)
110 GET#15,A$
```

ABLAUF EINES ASSEMBLER-UNTERPROGRAMMS: MEMORY EXECUTE

Diese Anweisung veranlaßt den Ablauf eines in Assembler-Sprache geschriebenen Unterprogramms und hat folgendes Format:

```
PRINT# lf,"M-E"adl/adh
```

Die Startadresse des Unterprogramms im Pufferspeicher der Diskette besteht aus zwei Bytes: Dem höherwertigen, adh, und dem niederwertigen, adl, Byte. Das ausgeführte Unterprogramm muß mit einer RETURN-Anweisung in folgender Form abgeschlossen sein:

```
RTS,$60
```

BENUTZERANWEISUNGEN: USER

Es stehen 10 besondere Benutzeranweisungen zur Verfügung, die in Tabelle 7-3 zusammengestellt sind. Die ersten beiden, U1 und U2, ersetzen die Anweisungen BLOCK READ und BLOCK WRITE; weitere sieben Anweisungen, U3 und U9, sind Sprungbefehle in Unterprogramme, während die zehnte Anweisung eine Einschalt-Routine anspricht. Anhang D enthält in der Tabelle für Speicherauszüge der Version 3 ROMs Angaben über die Routinen, in die U3 bis U9 springen. U1 und U2 werden in folgender Form benutzt:

```
PRINT# lf"Ux;sa,dr,t,s"
```

Hierbei ist x = 1 für U1 oder x = 2 für U2.

Tabelle 7-3. Benutzeranweisungen

Benutzer-Bezeichnung	Alternative Benutzer-Bezeichnung	Funktion
U1	UA	Ersatz für BLOCK-READ
U2	UB	Ersatz für BLOCK-WRITE
U3	UC	Springe nach $1300
U4	UD	Springe nach $1303
U5	UE	Springe nach $1306
U6	UF	Springe nach $D008
U7	UG	Springe nach $D00B
U8	UH	Springe nach $D00E
U9	UI	Springe nach $D0D5
U:	UJ	Einschalt-Routine $E18E

KAPITEL 8

CBM BASIC

Dieses Kapitel beschreibt die Syntax, d.h. den richtigen Gebrauch aller Anweisungen und Funktionen in CBM BASIC. Zuerst werden Anweisungen dem Alphabet nach beschrieben; dann folgt die Beschreibung der Funktionen.

Dieses Kapitel dient als Referenz für alle Anweisungen und Funktionen, wie sie in Kapitel 4, 5 und 6 zur Beschreibung von Konzepten der Programmerstellung und in den Programmbeispielen verwendet wurden.

DIREKTER DIALOG UND PROGRAMMFORM

Die meisten Befehle können sowohl in direktem Dialog als auch in Programmform verwendet werden. Sofern nicht anders vermerkt, können Sie davon ausgehen, daß ein Befehl in beiden Anwendungsformen anwendbar ist.

VERSIONEN VON CBM BASIC

Für alle Befehle und Funktionen wird angegeben, ob sie in allen Versionen von CBM BASIC vorkommen oder nur in CBM BASIC 4.0. Die Beschreibungen enthalten Querverweise zwischen Befehlen und Funktionen in CBM BASIC < 3.0 und CBM BASIC 4.0, soweit vorhanden. Für alle Befehle in BASIC 4.0 sind Disketten-Betriebssysteme der Version DOS 2.x erforderlich.

FORMATVEREINBARUNGEN

Befehle, Funktionen und Syntax werden in einem Format dargestellt, dessen Symbolik nachfolgend erklärt ist:

FORMATVEREINBARUNGEN

GROSSBUCHSTABEN	Formatangaben in Großbuchstaben geben die verbindliche Schreibung der Befehle und Funktionen wieder
kleinbuchstaben	Formatangaben in Kleinbuchstaben bezeichnen Angaben, die nach Anwendungsfall vom Benutzer eingesetzt werden
[ ]	Klammern kennzeichnen Angaben, die nicht zwingend erforderlich sind. Klammern erscheinen nicht bei der Anwendung der Befehle/ Funktionen.

In den Definitionen von Befehlen und Funktionen werden folgende Konstanten verwendet:

Tabelle 8-1. Gerätenummern

Gerätenummer	Gerät
0	Tastenfeld
1 (Ersatzwert)	Bandgerät #1
2	Bandgerät #2
3	Bildschirm
4	Drucker
5 - 7	Geräte am IEEE-Bus
8	Floppy Disk-Speicher
9 - 30	Geräte am IEEE-Bus
31 - 255	z. Zt. nicht verfügbar

Tabelle 8-2. Sekundäradressen

Gerät	Sekundär-adresse	Funktion
CBM-Band-geräte	0 +)	Öffnen zum Lesen
	1	Öffnen zum Schreiben
	2	Öffnen zum Schreiben mit Dateiende-Marke (EOF). Bandende-Marke (EOT) beim Schließen der Datei schreiben
CBM-Drucker	0 +)	Normaler Ausdruck
	1	Ausdruck nach Formatangaben
	2	Speichern von Formatangaben
	3	Speichern einer Zeilenzahl/Seite
	4	Freigabe von Fehlermeldungen
	5	Benutzer-entworfenes Zeichen speichern
	6	Zeilenabstände ändern
	7	Kleinschreibung
	8	Großschreibung
	9	Gerät 4 ausschalten
	10	Zurücksetzen
CBM Floppy Disk-Speicher	0	Nicht verwendet
	1	Nicht verwendet
	2 - 14	Öffnen zum Lesen/Schreiben wie angegeben
	15	Öffnen des Befehlskanals
+) Ersatzwert, d.h. bei fehlender Angabe verwendet		

BEFEHLE IN CBM BASIC

APPEND# BASIC 4.0

Format: APPEND#LD,"dateiname" [,Dlfw][ON Ugn]

Zweck: Anfügen von Daten an das Ende einer bereits bestehenden Sequentiellen Datei auf Diskette.

Bemerkung: Die Anweisung APPEND# öffnet eine bestehende Sequentielle Datei "dateiname" auf der Diskette in Laufwerk lfw und setzt den Adreßzeiger auf das Dateiende. Eine folgende PRINT#ld-Anweisung mit der gleichen logischen Dateinummer ld schreibt Eintragungen ab dem Dateiende weiter. Bei fehlender Laufwerknummer lfw wird Laufwerk 0 angenommen.

Beispiel:

```
DIREKTER DIALOG                          PROGRAMMFORM

APPEND#1,"DATEI"                         50 APPEND#1,"DATEI"
PRINT #1,A          *)                   60 PRINT #1,A

APPEND#3,"DATEI",D1                      50 APPEND#3,"DATEI",D1
PRINT #3,"ABC"      +)                   60 PRINT #3,"ABC"

*)INHALT DER VARIABLEN A  +)ZEICHENKETTE "ABC"
AN DAS DATEIENDE ANFUEGEN
```

BACKUP — BASIC 4.0

Format: BACKUP Dq TO Dz [ON Ugn]

Zweck: Duplizieren einer gesamten Diskette.

Bemerkung: Die Anweisung BACKUP dupliziert eine Quell-Diskette in Laufwerk Dq durch Übertragung ihres gesamten Speicherinhalts auf eine Ziel-Diskette in Laufwerk Dz. Siehe auch PRINT#DUPLICATE. Die Duplizierung einer gesamten Diskette dauert einige Minuten.

Beispiel:

```
DIREKTER DIALOG                          PROGRAMMFORM

BACKUP D0 TO D1
BACKUP D1 TO D0
```

Vorsicht: Alle Dateien auf der Quell-Diskette müssen vorschriftsmäßig geschlossen sein, ehe eine Duplizierung durchgeführt werden kann.

CLOSE

Format: CLOSE ld

Zweck: Schließen einer logischen Datei.

Bemerkung: Die Anweisung CLOSE ld schließt die logische Datei ld. Wenn ld nicht existiert, werden in BASIC < 3.0 alle logischen Dateien geschlossen, während in BASIC 4.0 eine Fehlermeldung erscheint. Nach Abschluß aller Dateizugriffe sollte jede Datei mit CLOSE geschlossen werden. Nach einer Schreiboperation löst CLOSE die Ausgabe des restlichen Speicherinhalts im Pufferspeicher aus. Für ausführliche Anwendungsbeispiele siehe Kapitel 6. Siehe auch DCLOSE.

Beispiel:

```
DIREKTER DIALOG                          PROGRAMMFORM
OPEN 1,4:CMD 1:LIST:PRINT#1:CLOSE 1      50 OPEN 1,4
                                         .
                                         .
                                         90 CLOSE 1
```

CLR

Format: CLR

Zweck: Rücksetzen aller numerischer Variablen auf Null, aller Zeichenketten-Variablen auf die leere Zeichenkette. Freigabe des von den Variablen belegten Speicherbereichs im Computer. Schließen aller im augenblicklichen Programm offenen logischen Dateien.

Bemerkung: Die Anweisung CLR hat die gleiche Wirkung wie das Aus/Einschalten des Computers und Neuladen des Programms. Ein Programm setzt seinen Ablauf nach Ausführung einer CLR-Anweisung fort, sofern die von CLR gelöschten Variablen den Ablauf nicht behindern.

Beispiel:

```
DIREKTER DIALOG                          PROGRAMMFORM

CLR                                      200 CLR
```

CMD

Format: CMD Id

Zweck: Offenhalten des Datenkanals zu einem externen Gerät.

Bemerkung: Die CMD-Anweisung sendet alle Ausgaben des Computers zum Drucker statt zum Bildschirm. Nach CMD folgende PRINT- und LIST-Anweisungen führen zu Ausdrucken statt zu Bildschirmdarstellungen. Die Wirkung von CMD wird durch eine PRINT#-Anweisung beendet. Siehe auch Kapitel 6.

Beispiel: Die folgende Eingabe in direktem Dialog benutzt die Anweisung CMD, um eine Programmauflistung auf dem Drucker auszugeben.

```
DIREKTER DIALOG

OPEN 1,4 : CMD 1 : DIRECTORY D1 : PRINT#1 : CLOSE 1

PROGRAMMFORM

20 OPEN 1,4 : CMD 1     (AUSGABE DES INHALTSVERZEICHNISSES
30 DIRECTORY D1          VON DISKETTE 1 AUF DEM DRUCKER)
40 PRINT#1 : CLOSE 1
```

COLLECT **BASIC 4.0**

Format: COLLECT [D Ifw] [ON Ugn]

Zweck: Löschen aller Blöcke auf einer Diskette, die nicht von im Inhaltsverzeichnis genannten Dateien belegt sind. Freigabe des Speicherplatzes

von Dateien, die geöffnet, aber nie mit OPEN geschlossen wurden. Neuanlegen des Verzeichnisses verfügbarer Blöcke (BAM).

Bemerkung: Die Speicherbelegung auf der Diskette in Laufwerk lfw wird überprüft. Fehlt die Angabe DIfw, dann wird die Diskette in Laufwerk 0 angenommen.

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

COLLECT
COLLECT D0
COLLECT D1
```

CONCAT

BASIC 4.0

Format: CONCAT [Dq,] "quelldatei" TO [Dz] "zieldatei" [ON Ugn]

Zweck: Verkettet Sequentielle Dateien.

Bemerkung. Der Inhalt einer Quelldatei auf der Diskette in Laufwerk Dq wird an das Ende einer Zieldatei auf der Diskette im Laufwerk Dz angefügt. Der Name der Quelldatei ändert sich nicht. Die Datei mit dem Namen "zieldatei" behält ihren ursprünglichen Inhalt, an den der Inhalt von "quelldatei" angefügt wird. Bei fehlenden Laufwerknummern q und/oder z wird Laufwerk 0 angenommen.

Beispiel:

```
DIREKTER DIALOG                          PROGRAMMFORM

CONCAT D1,"ABC" TO D0,"XYZ"  190 CONCAT "ABC" TO "XYZ"  *)
CONCAT "ABC" TO "XYZ"  *)
```

*)Beide Beispiele: die Datei "ABC" wird an das Ende von Datei "XYZ" angefügt. Beide Dateien befinden sich auf einer Diskette in Laufwerk 0.

Vorsicht: Nur Dateien, die geschlossen wurden, können verkettet werden.

CONT

Format: CONT

Zweck: Fortsetzung eines Programmablaufs nach Eingabe der Taste STOP oder nach Ausführung einer der Anweisungen STOP oder END.

Bemerkung: Die Anweisung CONT wird in direktem Dialog über das Tastenfeld eingegeben. Nach einer Programmunterbrechung mit der Meldung BREAK setzt CONT den Programmablauf an der Stelle der Unterbrechung fort.

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

CONT
```

COPY — BASIC 4.0

Format: COPY [Dq,] ["quelldatei"] TO [Dz,] ["zieldatei"] [ON Ugn]

Zweck: Kopieren einer Datei innerhalb des gleichen Floppy Disk-Speicher.

Bemerkung: Mit der Anweisung COPY kann eine einzelne Datei oder alle Dateien auf einer Diskette kopiert werden. Für COPY gibt es zwei Anwendungsformen: 1. Bei Fehlen der Dateinamen werden alle Dateien der Quell-Diskette Dq auf eine Ziel-Diskette Dz kopiert. 2. Eine "quelldatei" in Laufwerk Dq wird in eine "zieldatei" in Laufwerk dz kopiert. Bei fehlenden Angaben Dq, Dz wird Laufwerk 0 angenommen. Wenn der Name von "quelldatei" bereits auf der Diskette für die "zieldatei" besteht, dann wird der Kopiervorgang beendet und die Fehlermeldung FILE ALREADY EXISTS ausgegeben. COPY verändert keine der schon vorhandenen Dateien auf der Ziel-Diskette.

Beispiel: DIREKTER DIALOG — PROGRAMMFORM

```
COPY D1 TO D0
COPY D1,"ABC" TO D1,"XYZ"
COPY D1,"ABC" TO D0,"XYZ"
```

Vorsicht: Eine Datei muß geschlossen sein, ehe sie kopiert werden kann.

DATA

Format: DATA konstante[,konstante,konstante,konstante....]

Zweck: Speichern von numerischen Konstanten und Zeichenketten, die im weiteren Verlauf des Programms durch READ-Anweisungen gelesen werden können. Siehe auch READ.

Bemerkung: Die Anweisung(en) DATA kann an jeder Stelle im Programm stehen. Zeichenketten werden gewöhnlich zwischen Anführungszeichen gesetzt; Anführungszeichen erübrigen sich, wenn die Zeichenkette keine graphischen Symbole, Leerstellen, Kommas oder Semikolons enthält. Bei fehlenden Anführungszeichen werden Leerstellen, Kommas, Semikolons und graphische Symbole ignoriert. Anführungszeichen als Teil der Zeichenkette können nicht in DATA dargestellt werden; sie müssen mit der Funktion CHR$ (34) dargestellt werden. Die Anweisung DATA kann nicht in direktem Dialog verwendet werden.

Beispiel: DIREKTER DIALOG — PROGRAMMFORM

```
20 DATA ABC,"A,B,C,",123,1.23E-6
.
.
90 READ A$,B$,A%,A
```

DCLOSE — BASIC 4.0

Format: DCLOSE #Ld [ON Ugn]

Zweck: Schließen von Dateien auf Diskette.

Bemerkung: Die Anweisung DCLOSE schließt die logische Datei Id. Bei fehlender logischer Dateinummer werden alle augenblicklich offenen Dateien auf der Diskette geschlossen.

Beispiel:

```
DIREKTER DIALOG

DCLOSE
DCLOSE#1
DCLOSE ON U8
```

```
PROGRAMMFORM

310 DOPEN#1,"REL.DATEI",L50
 .
 .
460 DCLOSE#1
```

DEF FN

Format: DEF FN name (arg) = ausdruck

Zweck: Definieren und Benennen einer Benutzerfunktion.

Bemerkung: Die Anweisung DEF FN gibt dem Benutzer die Möglichkeit, eigene mathematische Funktionen zu definieren und in BASIC-Programmen zu benutzen. Die eigentliche mathematische Funktion wird durch ausdruck definiert, der aus jeder Verbindung von numerischen Konstanten, Variablen und/oder Operatoren bestehen kann. Die in ausdruck verwendeten Variablennamen sind im übrigen Programm wieder verwendbar, ohne mit DEF FN in Konflikt zu geraten. Eine DEF FN-Anweisung darf höchstens 80 Zeichen enthalten; durch Zitieren von anderen, mit DEF FN definierten Funktionen kann ein Benutzer Funktionen beliebiger Komplexität entwickeln.
Der eigentliche Name einer mit DEF definierten Funktion ist FNname (arg). Der Namenteil name folgt den Bildungsregeln für Variable und kann an anderer Stelle im Programm zur Definition einer weiteren Funktion wieder verwendet werden.
Die aus einer DEF FN-Anweisung hervorgehende Funktion kann anschließend im Programm in der Form FNname (arg) angesprochen werden. Nach Angabe eines Funktionsarguments arg liefert FN name (arg) den zugehörigen Funktionswert. Die DEF FN-Anweisung kann nicht in direktem Dialog verwendet werden. Wurde jedoch im Verlauf eines Programms mit DEF FN eine Benutzerfunktion definiert, dann ist ihre Benutzung in direktem Dialog nach Ende des Programmlaufs möglich. Bei Verwendung von Variablennamen für Zeichenketten und Ganzzahlen im Funktionsnamen name erscheint die Fehlermeldung TYPE MISMATCH. Wird auf die Funktion FNname zugegriffen, ehe ihre Definition durch DEF FN erfolgte, dann erscheint die Fehlermeldung UNDEFINED FUNCTION.

Beispiel:

```
DIREKTER DIALOG

?FNC(1)        *)
 3.14159264

*)NUR MOEGLICH NACH
  DEFINITION DER FUNKTION
  IN EINEM PROGRAMM
```

```
PROGRAMMFORM

20 DEF FNC(R)=3.1415926*R^2
 .
40 A=FNC(X)
 .
90 IF FNC(X)>60 THEN GOTO 200
```

DIM

Format: DIM var(sub)[,var(sub),. . .,var(sub)]

Zweck: Ankündigung des Umfangs einer im Programm verwendeten Tabelle zur vorsorglichen Reservierung von Speicherplatz.

Bemerkung: Die Angabe ein- oder mehrdimensionaler Tabellen (Felder) hat in einer DIM-Anweisung folgende Form:

var(sub_i)	1-dimensionale Tabelle
var(sub_i,sub_j)	2-dimensionale Tabelle
var(sub_i,sub_j,sub_k)	mehr-dimensionale Tabelle

Eine ausfühliche Beschreibung von Tabellen oder Feldern und ihrer Dimensionen finden Sie in Kapitel 4. Tabellen mit mehr als 11 Eintragungen (Felder mit mehr als 11 Elementen) müssen mit einer DIM-Anweisung angekündigt werden. Die DIM-Anweisung kann für ein-dimensionale Tabellen mit weniger als 11 Einträgen entfallen; für derartige Tabellen ist bereits Speicherplatz reserviert. Wird der angekündigte Tabellenumfang im Programm überschritten, dann erfolgt die Fehlermeldung BAD SUBSCRIPT. Fehlt eine DIM-Anweisung für eine Tabelle mit mehr als 11 Einträgen, dann erscheint die Fehlermeldung ?REDIM'ED ARRAY und das Programm wird abgebrochen. Eine DIM-Anweisung kann nach einer CLR-Anweisung im Programm wiederholt werden. Alle Eintragungen (Elemente) einer Tabelle werden mit DIM auf den Anfangswert 0 gesetzt.

Beispiel:

```
DIREKTER DIALOG                    PROGRAMMFORM

                                   20 DIM A(3)                    *)
                                   20 DIM A$(1,49)                #)
                                   20 DIM A(X),A$(Y,Z)            +)

*)1-SPALTIGE TABELLE FUER 4 EINTRAGUNGEN (0,1,2,3)
 VON GLEITKOMMAZAHLEN
#)2-SPALTIGE TABELLE (0,1) FUER 50 EINTRAGUNGEN
(0,1,...,49) VON ZEICHENKETTEN
+)DEN VARIABLEN X,Y,Z MUESSEN VOR AUSFUEHRUNG VON DIM
GANZZAHLEN ZUGEWIESEN SEIN
```

DIRECTORY **BASIC 4.0**

Format: DIRECTORY [Dlfw] [ON Ugn]

Zweck: Darstellen des Inhaltsverzeichnisses einer Diskette auf dem Bildschirm.

Bemerkung: Statt DIRECTORY kann auch das Wort CATALOG verwendet werden. Vergleichen Sie mit LOAD"$lfw". Bei fehlender Laufwerknummer Dlfw wird das Inhaltsverzeichnis beider Disketten ausgegeben. Bei leerem Laufwerk erscheint eine Fehlermeldung. Die DIRECTORY-Anweisung wird gewöhnlich in direktem Dialog verwendet. Die Auflistung des Inhaltsverzeichnisses kann mit der Taste STOP unterbrochen werden.

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

DIRECTORY
DIRECTORY D0
DIRECTORY D1
```

DLOAD — BASIC 4.0

Format: DLOAD "programmname" [D lfw] [ON Ugn]

Zweck: Laden eines BASIC-Programms von einer Diskette in den Speicher des Computers. Siehe auch LOAD.

Bemerkung: Die Anweisung DLOAD lädt das Programm "programmname" von der Diskette in Laufwerk Dlfw in den Speicher des Computers. Bei fehlender Laufwerknummer wird Laufwerk 0 angenommen. In BASIC 4.0 wird bei Eingabe der Tasten SHIFT und RUN/STOP das nächste Programm auf der Diskette in den Computer geladen und zum Ablauf gebracht.

Beispiel:

```
DIREKTER DIALOG                          PROGRAMMFORM

DLOAD "DATEI"                            10
DLOAD "DATEI",D1                         .
DLOAD "DATEI",D0 ON U8                   .      ← PROGR #1
                                         .
                                         200 DLOAD "PROGR #2"
```

DOPEN — BASIC 4.0

Format: DOPEN #ld, "dateiname" [Ly] [D lfw] [ON Ugn] [,W]

Zweck: Öffnen einer Datei für Daten für einen Schreib- und/oder Lese-Zugriff.

Bemerkung: Die Anweisung DOPEN öffnet eine Datei für Daten mit dem Namen „dateiname" unter der logischen Dateinummer lfw. Bei fehlender Laufwerknummer lfw wird Laufwerk 0 angenommen. Bei fehlender Aufzeichnungslänge Ly wird eine Sequentielle Datei angenommen. Eine Sequentielle Datei wird für einen Schreibzugriff geöffnet, wenn W angegeben ist. Relative Dateien werden für Lese- oder Schreib-Zugriffe geöffnet, so daß der Parameter W entfallen kann.

Beispiel:

```
DIREKTER DIALOG                  PROGRAMMFORM

DOPEN#1,"DATEI"                  50 DOPEN#1,"DATEI"
DOPEN#1,"DATEI"L50,D1            50 DOPEN#1,"DATEI"L50,D1      *)

*)OEFFNEN DER RELATIVEN DATEI "DATEI" AUF DISKETTE 1
FUER EINEN LESE/SCHREIBZUGRIFF. JEDE AUFZEICHNUNG AUF
DER DATEI IST 50 ZEICHEN LANG.
```

DSAVE — BASIC 4.0

Format: DSAVE "programmname" [, D lfw] [ON Ugn]

Zweck: Speichern eines Programms im Computer auf einer Diskette.

Beispiel:

```
DIREKTER DIALOG                                PROGRAMMFORM

DSAVE "PROGRAMM"
DSAVE "PROGRAMM",D1
```

END

Format: END

Zweck: Beenden eines Programmlaufs und Rückkehren zum direkten Dialog.

Bemerkung: Die Anweisung END kann ein Programm mit einem oder mehreren Abschlußpunkten versehen, die nicht mit dem physikalischen Ende des Programms übereinstimmen. Befinden sich mehrere Programme im Speicher des Computers, dann kann jedes Programm mit END abgeschlossen werden. Siehe in diesem Zusammenhang die Anwendung von RUN. Die END-Anweisung wird nur in Programmform verwendet.

Beispiel:

```
DIREKTER DIALOG                                PROGRAMMFORM

                                               2000 END
```

FOR-NEXT-STEP

Format:

```
FOR i = start TO end STEP increment
  .
  .  <------ Schleifenanweisungen
  .
NEXT i
```

mit:

i	Schleifenindex
start	Startwert des Schleifenindex
end	Endwert des Schleifenindex
increment	Schrittweite der Schleifenindexänderungen

Zweck: Wiederholte Ausführung von Anweisungen in gezählten Schleifendurchläufen.

Bemerkung: Alle Anweisungen zwischen einer FOR- und NEXT-Anweisung werden gleich häufig und wiederholt ausgeführt. Der Schleifenindex i kann in der Anweisung NEXT angegeben werden. Eine einzige NEXT-Anweisung ist zulässig, wenn verschachtelte Schleifen in der gleichen Programmzeile enden. Die NEXT-Anweisung hat dann folgende Form:

NEXT $i_1, i_2, i_3 \ldots$

Jede FOR-NEXT-Schleife wird mindestens einmal ausgeführt, selbst wenn der Startwert start größer als der Endwert end ist. Startwert, Endwert und Schrittweise des Schleifenindex i werden nur einmal bei der ersten Ausführung einer FOR-NEXT-Anweisung gelesen. Sie können nicht innerhalb der Schleife geändert werden. Lediglich der Schleifenindex i kann innerhalb der Schleife geändert werden. Hierdurch kann der Abbruch für Schleifendurchläufe von Bedingungen

innerhalb der Schleife abhängig gemacht werden: Setzen Sie i auf den Endwert, und die Schleifendurchläufe sind beendet. Springen Sie nicht mit GOTO aus einer FOR-NEXT-Schleife. Vermeiden Sie eine END-Anweisung innerhalb der Schleife. FOR-NEXT-Schleifen können verschachtelt werden. Jede verschachtelte Schleife muß einen Schleifenindex mit eigenem Namen haben. Jede verschachtelte Schleife muß vollständig innerhalb einer äußeren Schleife enthalten sein; die Schleifen dürfen höchstens am gleichen Programmpunkt enden.

Beispiel:

```
DIREKTER DIALOG   PROGRAMMFORM

                  10 FOR I=1 TO 100
                   .   ← SCHLEIFENANWEISUNGEN
                  90 NEXT I

                  10 FOR I=A+14 TO B-64+C/2 STEP 0.25
                   .   ← SCHLEIFENANWEISUNGEN
                  90 NEXT I

                  10 FOR I=100 TO 1 STEP -1
                   .   ← ANWEISUNGEN DER AEUSSEREN SCHLEIFE
                  50 FOR J=1 TO 100
                   .   ← ANWEISUNGEN DER INNEREN SCHLEIFE
                  90 NEXT I,J
```

GET

Format: GET var

Zweck: Lesen eines einzelnen Zeichens vom Tastenfeld und Zuweisen zu einer Variablen.
Bei Ausführung einer GET-Anweisung wird das nächste Zeichen aus dem Tastenfeld-Speicher gelesen und der Variablen var zugewiesen. Ist der Tastenfeldspeicher leer, dann werden der Variablen var je nach Variablentyp der Zahlenwert 0 oder die leere Zeichenkette zugewiesen. GET wird zum Lesen von Antworten durch Eingabe eines einzelnen Zeichens am Tastenfeld verwendet. GET kann auch die Tasteneingabe RETURN lesen, die in Form der Variablen CHR$ (13) der Variablen var zugewiesen wird (siehe Anhang A).

Stellt var eine numerische Variable dar, dann wird bei fehlender Tasteneingabe der Zahlenwert 0 in var gespeichert. Der gleiche Tastenwert 0 wird jedoch auch bei der Tasteneingabe 0 gespeichert.

Ist var eine numerische Variable und wird nicht eine der Zahlen 0 bis 9 eingegeben, dann erfolgt eine Fehlermeldung ? SYNTAX ERROR und das Programm wird unterbrochen.

Die GET-Anweisung kann mehr als eine Variable in ihrer Parameterliste enthalten; die Anwendung derartiger GET-Anweisungen ist jedoch schwierig:

GET var,var,. . .,var

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM
                         10 GET C$
                         10 GET A,B,C
                         10 GET A$:IF A$="" THEN GOTO 10
```

GET#

Format: GET #ld, var

Zweck: Lesen eines einzelnen Zeichens von dem Speicher eines externen Geräts.

Bemerkung: Die GET#-Anweisung kann nur in einem Programm verwendet werden. GET# liest ein einzelnes Zeichen von einem externen Gerät und weist das Zeichen der Variablen var zu. Das externe Gerät wird mit der logischen Dateinummer ld identifiziert. Die logische Dateinummer ld muß mit der Angabe in einer früheren OPEN- oder DOPEN-Anweisung übereinstimmen. Die Anweisungen GET# und GET behandeln Variable und Eingabedaten identisch.

Beispiel:

```
DIREKTER DIALOG                    PROGRAMMFORM

                                   10 OPEN 1,1,0
                                    .
                                   50 GET#1,X$   : PRINT X$;
                                    .
```

GOSUB . . . RETURN

Format: GOSUB zn

Zweck: Verzweigen zu und Zurückkehren von einem Unterprogramm.

Bemerkung: Eine GOSUB-Anweisung verzweigt zu einem Unterprogramm mit der ersten Zeilennummer zn und kehrt aus dem Unterprogramm mit der Anweisung RETURN zu der der Anweisung GOSUB folgenden Programmzeile zurück. Eine GOSUB-Anweisung kann an jeder Stelle in einem Programm erfolgen; ein Unterprogramm kann folglich von jeder Stelle in einem Programm aus aufgerufen werden. Unterprogramme können verschachtelt sein; das bedeutet, daß von einem Unterprogramm aus andere Unterprogramme aufgerufen werden können. In CBM BASIC sind 26 Verschachtelungen von Unterprogramm-Aufrufen zulässig; das bedeutet, daß 25 GOSUB-Anweisungen ausgeführt werden können, ehe die erste RETURN-Anweisung erfolgt.

Beispiel:

```
DIREKTER DIALOG   PROGRAMMFORM

                  100 GOSUB 2000   ← SPRUNG INS UNTERPROGRAMM
                  110 A=SIN(...    ← RUECKSPRUNG AUS DEM
                                     UNTERPROGRAMM
                  .
                  .
                  2000             ←
                    .              ← UNTERPROGRAMM
                    .              ←
                  2100 RETURN      ←
```

GOTO

Format: GOTO zn

Zweck: Unbedingter Sprung zu einer angegebenen Programmzeile.

Bemerkung: Eine GOTO-Anweisung löst in einem Programmablauf den Sprung zur Programmzeile mit der Zeilennummer zn aus. Bei Eingabe einer GOTO-Anweisung in direktem Dialog erfolgt ein Sprung in die Programmzeile eines im Computer gespeicherten Programms mit der angegebenen Zeilennummer zn.

```
DIREKTER DIALOG                PROGRAMMFORM
GOTO 90   *)                   100 GOTO 200
*)SPRUNG ZU PROGRAMMZEILE 90 EINES IM COMPUTER
GESPEICHERTEN PROGRAMMS
```

HEADER

BASIC 4.0

Format: HEADER "diskname", D lfw [,Ivv] [ON Ugn]

Zweck: Formatieren einer fabrikneuen Diskette oder Wiederholung der Formatierung einer alten Diskette.

Bemerkung: Eine HEADER-Anweisung formatiert eine Diskette, die noch nicht in einem FLOPPY DISK-Speicher in Gebrauch war; angewandt auf eine bereits benutzte Diskette können mit HEADER alle Dateieintragungen und das Inhaltsverzeichnis gelöscht und die Formatierung wiederholt werden. Bei Formatierung mit der Anweisung HEADER werden auf der Oberfläche der Diskette Sektoren (Blöcke) auf jeder Spur markiert und das Inhaltsverzeichnis sowie das Verzeichnis verfügbarer Blöcke (BAM) initiiert. Die zu formatierende Diskette muß sich in Laufwerk lfw befinden. Der Diskette wird der Name "diskname" und die Kennzahl vv gegeben. Diese Angaben erscheinen in Negativschrift als Kopfeintrag im Inhaltsverzeichnis. Die HEADER-Anweisung wird gewöhnlich in direktem Dialog ausgeführt. Da die Änderungen durch HEADER auf einer benutzten Diskette nicht rückgängig zu machen sind, muß diese Anweisung mit Vorsicht verwendet werden. Vor ihrer Ausführung erscheint daher auf dem Bildschirm die letzte Rückfrage ARE YOU SURE? (Sind Sie sicher?). Erst nach Eingabe von Y für YES und der Taste RETURN erfolgt die Ausführung.

Wurde die Diskette nicht in das Laufwerk geelegt, oder eine schreibgeschützte Diskette oder eine Diskette mit defekter magnetischer Oberfläche geladen, dann erfolgt die Fehlermeldung ? BAD DISK.

Beispiel:

```
DIREKTER DIALOG                                PROGRAMMFORM

HEADER "DISK#1",D0,I02    *)

   *) FORMATIEREN DER DISKETTE 0 UND VERSEHEN MIT NAME
      (DISK#1) UND KENNZAHL (02)
```

IF-THEN

Format:

IF bedingung THEN anweisung [: anweisung . . .]
Bedingte Ausführung von Anweisungen

IF bedingung THEN zn
IF bedingung GOTO zn
} Bedingter Sprung

Zweck: Bedingter Sprung zu einer angegebenen Programmzeile oder bedingte Ausführung einer oder mehrerer Anweisungen.

Bemerkung: Trifft die nach IF angegebene Bedingung zu, dann wird die nach THEN angegebene anweisung (oder Anweisungen) ausgeführt. Trifft die angegebene bedingung nicht zu, dann setzt der Programmablauf mit der nächsten Programmzeile fort und die nach THEN angegebene anweisung (oder Anweisungen) wird nicht ausgeführt. Für einen bedingten Sprung mit IF-THEN sind folgende Formen für die Angabe der Zeilennummer, zu der gesprungen wird, zulässig:

```
IF A = 1 THEN 50
IF A = 1 GOTO 50        } gleichwertig
IF A = 1 THEN GOTO 50
```

Wenn ein unbedingter Sprung in Form der Anweisung GOTO eine von mehreren Anweisungen nach THEN bildet, dann muß die Sprunganweisung die letzte Anweisung in der Programmzeile sein und die Form GOTO zn haben. Wird diese Reihenfolge nicht eingehalten, dann werden Anweisungen nach GOTO nie ausgeführt.
In direktem Dialog können folgende Anweisungen nicht in einer IF-THEN-Anweisung verwendet werden: DATA, GET, GET#, INPUT, INPUT#, REM, RETURN, END, STOP, WAIT.

In einem Programm darf die IF-THEN-Anweisung nicht die Anweisungen CONT und DATA enthalten. Bildet eine FOR-NEXT-Schleife eine der Anweisungen nach THEN, dann muß die vollständige FOR-NEXT-Schleife auf der Programmzeile von IF-THEN stehen. Ebenso können weitere IF-THEN-Anweisungen nach THEN folgen, solange sie auf der gleichen Programmzeile enthalten sind, wie das erste IF-THEN. Statt verschachtelter IF-THEN-Anweisungen werden jedoch Logische Operatoren bevorzugt. Ein Beispiel: die beiden folgenden Anweisungen sind gleichwertig, jedoch wird die Form der zweiten bevorzugt:

```
10 IF A$ = "X" THEN IF B = 2 THEN IF C > D THEN 50
10 IF A$ = "X" AND B = 2 AND C > D THEN 50
```

Zeilennummern, die in den Anweisungen nach THEN genannt werden, müssen im gleichen Programm auch tatsächlich vorhanden sein.

Beispiel:

DIREKTER DIALOG

PROGRAMMFORM

```
10 IF X>Y THEN A=1.5
10 IF X>Y THEN A=1.5 : GOTO 300
```

INPUT

Format:

INPUT var [, var, . . ., var]

INPUT "mitteilung"; var [, var, . . ., var]

Zweck: Möglichkeit für Benutzereingaben während eines Programmablaufs.

Bemerkung: Die Anweisung INPUT kann nur in Programmen verwendet werden. Bei Ausführung der Anweisung INPUT wird ein Fragezeichen auf dem Bildschirm ausgegeben, das den Benutzer zur Dateneingabe auffordert.

Eingegebene Daten müssen in Anzahl und Typ mit den Angaben in der Parameterliste der INPUT-Anweisung übereinstimmen. Bei mehr als einer Variablen var müssen die Eingaben des Benutzers durch Komma getrennt sein.

Eine "mitteilung" an den Benutzer wird jeweils vor dem Fragezeichen auf dem Bildschirm dargestellt. Eine "mitteilung" kann aus bis zu 80 Zeichen bestehen.

Werden mehr als eine, aber weniger als die erforderliche Anzahl von Daten eingegeben, dann meldet sich INPUT mit zwei Fragezeichen (??), bis die erforderliche Anzahl von Benutzereingaben erfüllt ist. Werden zuviele Daten eingegeben, dann erscheint die Nachricht #EXTRA IGNORED: Überzählige Eingaben werden ignoriert und das Programm fortgesetzt. Für numerische Variable var dürfen nur Zahlen eingegeben werden; für Zeichenketten-Variable var können sowohl Zeichenketten wie Zahlen eingegeben werden.

Beispiel:

```
ANWEISUNG                    BENUTZER-EINGABE:  RESULTIERENDE
                                                ZUWEISUNGEN:

20 INPUT A,B,C$              ?123,456,ABC       A=123,B=456,C$=ABC
30 INPUT A                   ?ABC
                             ?REDO FROM START
                             ?123               A=123
40 INPUT "NAME:";N$          NAME:? NATALIE     N$=NATALIE
```

Vorsicht: Wird die Taste RETURN bedient, ohne daß Daten als Antwort auf eine INPUT-Anweisung eingegeben wurden, dann wird der Programmlauf abgebrochen und der Computer kehrt zum direkten Dialog zurück. Der Programmablauf kann dann durch Eingabe von CONT fortgesetzt werden.

INPUT#

Format: INPUT #ld var [,var, . . ., var]

Zweck: Lesen von Daten aus einem externen Speicher und Zuweisen zu Variablen.

Bemerkung: Die INPUT#-Anweisung ist nur in Programmen verwendbar. Eine INPUT#-Anweisung liest Daten von einem externen Speicher, der unter der logischen Dateinummer ld geöffnet wurde, und weist die gelesenen Daten den Variablen var in der Parameterliste von INPUT# zu. Die gelesenen Daten müssen in Anzahl und Typ mit den Angaben in der Parameterliste von INPUT# übereinstimmen.
Wird das Endezeichen einer Aufzeichnung entdeckt, noch ehe allen Variablen var in der Parameterliste von INPUT# Daten zugewiesen werden konnten, dann wird der Fehler-Status OUT OF DATA erzeugt, aber der Programmlauf nicht unterbrochen.

Die Anweisungen INPUT# und INPUT unterscheiden sich nur darin,

daß INPUT# seine Eingabe von einer logischen Datei erhält. Außerdem stellt INPUT# keine Fehlermeldungen auf dem Bildschirm dar; stattdessen meldet es Fehler in Form einer Statusvariablen, die vom Programm abgefragt werden muß.

Die Kapazität des Pufferspeichers von 80 Zeichen beschränkt die Länge von Zeichenketten, die mit INPUT# gelesen werden können, auf 80 Zeichen (79 Zeichen und einen Wagenrücklauf).

Die Anweisung INPUT# behandelt Kommas und das Symbol für Wagenrücklauf als Trennsymbole zwischen Daten; diese Symbole werden nicht als Daten gelesen.

Beispiel:

```
DIREKTER DIALOG                                  PROGRAMMFORM

                                                 10 INPUT#1,A
                                                 10 INPUT#1,A$
                                                 10 INPUT#1,A$,A    *)

  *)VON DER LOGISCHEN DATEI#1 WERDEN 2 DATEN GELESEN.
    ALS ERSTES DATUM WIRD EINE ZEICHENKETTE ERWARTET UND
    DER VARIABLEN A$ ZUGEWIESEN.
```

LET

Format:

```
    var = wert
LET var = wert
```

Zweck: Zuweisen eines Zahlenwertes oder einer Zeichenkette zu einer Variablen.

Bemerkung: Die Verwendung des Wortes LET ist freigestellt. Der Variablen var wird ein Wert zugewiesen, der auch das Ergebnis der Berechnung eines Ausdrucks sein kann.

Beispiel:

```
DIREKTER DIALOG                  PROGRAMMFORM

A=SQR(789)+SIN(0.5)              10 A=SQR(X)+SIN(Y)
A$="* TITEL *"                   10 A$="** TITEL **"
                                 10 M(0,0)=100
                                 10 M$(I,J,K,L)="** TITEL **"
```

LIST

Format:

```
LIST                LIST – zn
LIST zn             LIST zn –
LIST zn1 – zn2
```

Zweck: Ausgabe eines im Computer gespeicherten Programms (oder von Teilen hiervon) durch Darstellung auf einem Bildschirm oder Drucker.

Bemerkung: Zur Programmdarstellung auf dem Bildschirm wird die Anweisung LIST eingegeben. Mit den Zeilennummern zn können einzelne Zeilen oder Programmausschnitte zur Darstellung ausgewählt werden.

Zur Programmdarstellung auf einem Drucker muß über die Anweisungen OPEN und CMD der Ausgabekanal des Computers vom Bildschirm

auf den Drucker umgeschaltet werden.

Eine Programmauflistung ist mit folgenden Formatänderungen verbunden:

1. Die Abkürzung ? für PRINT wird ausgeschrieben als PRINT:

```
?A          wird zu      PRINT A
```

2. Leerstellen vor Zeilennummern werden unterdrückt:

```
 50 A=1       wird zu    50 A=1
100 A=A+1     wird zu   100 A=A+1
```

3. Zwischen Zeilennummer und Anfang der Anweisung wird eine Leerstelle eingefügt:

```
55A=B-2      wird zu     55 A=B-2
```

4. Die Darstellung von Programmzeilen beginnt mit Spalte 2 statt Spalte 1.

Eine LIST-Anweisung wird immer in direktem Dialog verwendet. Bei Verwendung von LIST in einem Programm erfolgt zwar die Programmauflistung, der Programmlauf wird aber unterbrochen und der Computer kehrt zum direkten Dialog zurück. Beim Versuch, den Programmablauf mit CONT fortzusetzen, wird lediglich die Programmauflistung durch LIST wiederholt.

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

OPEN1,4 : CMD1 : LIST : PRINT#1 : CLOSE1
```

LOAD

Format: LOAD ["programmname"] [,gn] **BANDGERÄTE**

Zweck: Laden eines Programms von der Kassette eines Bandgeräts in den Computer.

Bemerkung: Die Anweisung LOAD lädt das Programm mit dem Namen "programmname" von der Kassette in Bandgerät #gn in den Computer.

In direktem Dialog ausgeführt, schließt die Anweisung LOAD alle offenen logischen Dateien und löscht Variable und Programmzeilen eines im Computer gespeicherten Programms, ehe das bezeichnete Programm geladen wird.

In Programmform ausgeführt, läßt die Anweisung LOAD alle offenen logischen Dateien offen und veranlaßt den Ablauf des eben geladenen Programms. In diesem Fall ermöglicht LOAD die Verkettung von Programmabläufen.

Bei fehlender Gerätenummer gn wird Bandgerät #1 angenommen; bei fehlendem Namen "programmname" wird das nächste, auf dem Band angetroffene Programm geladen.

Beispiel:

```
DIREKTER DIALOG

LOAD                  : LADEN DES NAECHSTEN PROGRAMMS AUF
                        BANDGERAET#1
LOAD "",2             : LADEN DES NAECHSTEN PROGRAMMS AUF
                        BANDGERAET#2
LOAD "PROGR#1"        : SUCHEN UND LADEN DES PROGRAMMS
                        "PROGR#1" AUF BANDGERAET#1
N$="PROGR#1"
LOAD N$               : WIE OBEN
```

Format: LOAD "lfw: programmname", gn **FLOPPY-DISK-SPEICHER**

Zweck: Laden eines Programms von der Diskette eines Floppy-Disk-Speichers in den Computer.

Bemerkung: Die LOAD-Anweisung lädt das Programm mit dem Namen "programmname" von der Diskette in Laufwerk lfw in den Computer. Die Gerätenummer gn hat für CBM Computer-Systeme den Standardwert 8. Fehlt die Gerätenummer gn, dann ist automatisch der Ersatzwert 1, d.h. Bandgerät # 1 angesprochen. Bei einem Sternsymbol statt des Programmnamens wird das erste Programm auf der bezeichneten Diskette in den Computer geladen.

Beispiel:

```
DIREKTER DIALOG

LOAD "0:*",8
LOAD "0:PROGR#1",8

A$="0:PROGR#1"
LOAD A$,8
```

```
PROGRAMMFORM

10
 .
 .              ← PROGR#1
 .
500 LOAD "0:PROGR#2",8
```

INHALTSVERZEICHNIS EINER DISKETTE LADEN **BASIC < 3.0**

Zur Darstellung des Speicherverzeichnisses einer Diskette auf dem Bildschirm dient in BASIC 4.0 die Anweisung DIRECTORY. In BASIC < 3.0 muß mit der Anweisung LOAD eine Datei mit dem Namen $0 (für die Diskette in Laufwerk 0) oder $1 (für die Diskette in Laufwerk 1) geladen werden.

Beispiel:

```
DIREKTER DIALOG                                  PROGRAMMFORM

LOAD "$0",8          ← BENUTZEREINGABE

SEARCHING FOR $0
LOADING
READY.

LIST                 ← BENUTZEREINGABE
```

NEW

Format: NEW

Zweck: Löschen eines augenblicklich im Computer gespeicherten Programms vor Eingabe eines neuen Programms.

Bemerkung: Die Anweisung NEW löscht nicht nur die Variablen und Tabellen (Felder) eines Programms, wie die Anweisung CLR, sondern löscht auch das im Computer gespeicherte BASIC-Programm. Vor Eingabe eines neuen Programms über das Tastenfeld sollten alte Programmspeicherungen mit der Anweisung NEW gelöscht werden, um Überlagerungen von Programmen zu vermeiden.

Die Anweisung NEW wird ausschließlich in direktem Dialog eingegeben. In einem Programm verwendet, führt NEW zur Selbstzerstörung des Programms.

Beispiel:

```
DIREKTER DIALOG                    PROGRAMMFORM

NEW
```

ON . . . GOSUB

Format: ON ausdruck GOSUB zn_1 [,zn_2, . . ., zn_n]

Zweck: Bedingte Verzweigung zu einem von mehreren Unterprogrammen in einem Hauptprogramm.

Bemerkung: Die Bedingung ist in der Formulierung eines arithmetischen oder logischen Ausdrucks enthalten, dessen Auswertung einen der Zahlenwerte 1, 2, 3 . . . liefert und damit entscheidet, zu welchem der Unterprogramme zn_1, zn_2, zn_3 . . . verzweigt werden soll, wobei zn die erste Zeile des jeweiligen Unterprogramms bezeichnet.

Nach Rückkehr aus einem Unterprogramm setzt der Ablauf des Hauptprogramms mit der Anweisung unmittelbar nach der ON . . . GOSUB-Anweisung fort.

Liefert die Auswertung von ausdruck eine negative Zahl, dann erfolgt die Fehlermeldung "ILLEGAL QUANTITY". Ist der Zahlenwert 0 oder größer als die Anzahl von Zeilennummern zn in der GOSUB-Liste, dann wird die ON . . . GOSUB-Anweisung übersprungen.

ON . . . GOSUB wird üblicherweise in Programmform ausgeführt. Eine Ausführung in direktem Dialog ist nur möglich, wenn die Verzweigung zu einem im Computer gespeicherten Programm erfolgen kann.

Beispiel:

```
DIREKTER DIALOG                    PROGRAMMFORM

                                   50 ON A GOSUB 100,200,300
```

ON . . . GOTO

Format: ON ausdruck GOTO zn_1 [,zn_2, . . ., zn_n]

Zweck: Bedingte Verzweigung zu einer von mehreren Zeilen des Hauptprogramms.

Bemerkung: Format und Funktion einer ON . . . GOTO-Anweisung sind analog zur Anweisung ON . . . GOSUB.

Beispiel:

```
DIREKTER DIALOG                    PROGRAMMFORM

                                   50 X=X+1
                                   60 ON X GOTO 500,600,700
```

OPEN

Format: OPEN ld [,gn] [,sa] [,"dateiname"] BANDGERÄTE

Zweck: Öffnen einer logischen Datei zur Vorbereitung des Datenverkehrs mit einem externen Gerät. Vergleiche DOPEN.

Bemerkung: Die logische Datei ld öffnet den Datenverkehr zum externen Gerät mit der Gerätenummer gn für einen Zugriff, der mit der Sekundäradresse sa bezeichnet ist. Der Zugriff erfolgt auf eine Banddatei mit dem Namen "dateiname".
Bei fehlender Gerätenummer gn wird das Bandgerät # 1 angesprochen. Bei fehlender Sekundäradresse sa wird die Datei gelesen. Bei fehlendem Dateinamen erfolgt der Zugriff auf die nächste, auf dem Band angetroffene Datei. Mit der Sekundäradresse sa=1 erfolgt der Zugriff auf die Datei zum Schreiben, mit sa=2 wird am Ende des Schreib-Zugriffs beim Schließen der Datei mit CLOSE eine Bandende-Marke geschrieben.
Die logische Dateinummer ld ist auf den Wertebereich 1 bis 255 beschränkt. Mit einer OPEN-Anweisung kann der Datenverkehr zu einem Bandgerät (Gerätenummer gn=1 oder 2), Floppy-Disk-Speicher (Gerätenummer gn=8), Drucker (Gerätenummer gn=4) oder Bildschirm (Gerätenummer gn=3) geöffnet werden.

Beispiel:

```
DIREKTER DIALOG                        PROGRAMMFORM

OPEN 1                        *)       10 OPEN 1
OPEN 1,1                      *)       10 OPEN 1,1
OPEN 1,1,0,"PROGR#1"          +)       10 OPEN 1,1,0,"PROGR#1"
OPEN 1,1,2                    #)       10 OPEN 1,1,2
OPEN 1,1,2,"PROGR#1"          &)       10 OPEN 1,1,2,"PROGR#1"

OEFFNEN DER LOG.DATEI#1 AUF BANDGERAET#1 ZUM LESEN
 *)DER NAECHSTEN,AUF BAND GEFUNDENEN DATEI  +)DER DATEI
   MIT NAMEN "PROGR#1"

OEFFNEN DER LOG.DATEI#1 ZUM SCHREIBEN MIT ABSCHLIESSENDER
  BANDENDE-MARKE (EOT). DIE AUFZEICHNUNG BEGINNT AN DER
  MOMENTANEN BANDSTELLUNG. #)SCHREIBEN OHNE BENENNUNG
  DER DATEI  &)DIE DATEI ERHAELT DEN NAMEN "PROGR#1"
```

Format: OPEN ld, gn, sa, "lfw: dateiname, typ [,zugriff]" FLOPPY DISK-SPEICHER

Bemerkung: Der Zugriff auf eine Diskette in einem Floppy Disk-Speicher erfordert die Angabe der Laufwerknummer lfw=0 oder 1. Mit parametertyp wird der Typ der Datei angegeben, auf die der Zugriff erfolgen soll: typ = P für eine Programm-Datei, typ = U für eine Datei mit wahlfreiem Zugriff und typ = SEQ für eine Sequentielle Datei. Sequentielle Dateien werden bei fehlendem Parameterzugriff zum Lesen geöffnet; zum

Schreiben in die Datei ist die Angabe zugriff = WRITE (oder W) erforderlich.
Eine bestehende Sequentielle Datei kann zum Schreiben geöffnet werden, wenn der Laufwerknummer lfw das Symbol @ vorangesetzt ist.
Der vorhandene Inhalt der Sequentiellen Datei wird hierbei überschrieben.
Die Gerätenummer gn muß angegeben werden; bei fehlender Angabe wird automatisch mit dem Bandgerät gearbeitet.

Beispiel:

```
DIREKTER DIALOG                          PROGRAMMFORM

*)OPEN 1,8,2,"1:DAT,SEQ,READ"    10 OPEN 1,8,2,"0:DAT,SEQ,READ"
+)OPEN 1,8,3,"1:DAT,SEQ,WRITE"   10 OPEN 1,8,3,"1:DAT,SEQ,WRITE"
#)OPEN 1,8,4,"@1:DAT,SEQ,WRITE" 10 OPEN 1,8,4,"@1:DAT,SEQ,WRITE"
  OEFFNEN DER LOG.DATEI#1 AUF DISKETTE#1 ZUM
  *)LESEN DER DATEI "DAT"
  +)ZUM SCHREIBEN IN DIE NEUE DATEI "DAT"
  #)ZUM UEBERSCHREIBEN DER ALTEN DATEI "DAT"
```

POKE

Format: POKE speicheradresse, byte

Zweck: Direktes Schreiben von Daten in einen Speicherplatz des Computers.

Bemerkung: Das gespeicherte Datum kann eine Zahl zwischen 0 und 255 sein. Mit diesen Zahlen kann das Zeichenrepertoire des CBM-Computers kodiert werden (siehe Anhang A, Tabelle A-4).

Beispiel:

```
DIREKTER DIALOG                          PROGRAMMFORM

POKE 32768,193                           20 POKE 32768,ASC("A")-64
```

PRINT

Format: PRINT daten [x daten . . . x daten]
oder: ? daten [x daten . . . x daten]
x = "," oder x = ";"

Zweck: Datenausgabe vom Computer zum Bildschirm oder zum Drucker.

Bemerkung: Die Anweisung PRINT kann alternativ durch ein Fragezeichen (?) dargestellt werden. Die Elemente daten der Parameterliste in PRINT können Zahlen, Zeichenketten oder Namen von Variablen sein, deren Inhalt von PRINT dargestellt wird. Die Anordnung der daten in der Darstellung (auf dem Bildschirm oder Drucker) wird von den Trennsymbolen Komma und Semikolon bestimmt.

Darstellungsfelder: Für Zahlen im Bereich 0.01 bis 999999999 wird die gewohnte Schreibweise verwendet; für Zahlen außerhalb dieses Bereichs wird die wissenschaftliche Schreibweise verwendet. Das Darstellungsfeld für Zahlen hat folgende Form:

Zeichenketten werden in der Form dargestellt, in der sie in der Parameterliste von PRINT stehen. Zeichenketten müssen von Anführungszeichen eingeschlossen sein.

Darstellungsformate: Das erste Element der Parameterliste wird an der augenblicklichen Stellung des Kursors dargestellt. (Siehe TAB für die Veränderung der Kursorposition). Die Stellung des folgenden Elements daten in der Darstellung hängt von der Art des Trennsymbols (Komma oder Semikolon) ab.
Ein Semikolon bewirkt, daß ein Element daten unmittelbar anschließend an das vorangegangene Element dargestellt wird.
Ein Komma bewirkt, daß das nachfolgende Element daten an der nächstliegenden Tabulator-Position dargestellt wird. Diese Tabulator-Positionen sind bei einem 40spaltigen Bildschirm die Spalten 1, 11, 21 und 31 (mit Fortsetzung bei Spalten 41, 51, 61 und 71 für einen 80-spaltigen Bildschirm). Ein Komma vor dem ersten Element daten bewirkt, daß der Kursor um eine Tabulator-Position vorrückt, ehe die Darstellung des Elements beginnt. Fehlen Komma oder Semikolon am Ende der Parameterliste, dann wird nach Darstellung des letzten Elements daten ein Zeilensprung (Wagenrücklauf beim Drucker) ausgelöst.

Beispiel:

```
DIREKTER DIALOG                PROGRAMMFORM

PRINT 1,2                      50 PRINT 1,2,"AB","CD"
 1          2                  50 PRINT 1;2;"AB";"CD"
                               50 PRINT,1;2;"AB";"CD"
PRINT 1;2;"AB";"CD"
 1  2 ABCD                     50 PRINT 1,
                               60 PRINT 2 WIRKT WIE: PRINT 1,2
PRINT , 1;2;"AB";"CD"          50 PRINT 1;
            1  2 ABCD          60 PRINT 2 WIRKT WIE: PRINT 1;2
                               50 PRINT 1
                               60 PRINT 2 WIRKT WIE: PRINT 1:PRINT 2

PRINT 1 : PRINT 2 : PRINT "AB"
 1
 2
AB
```

PRINT#

Format: PRINT #Id,daten; c$; daten; c$; . . . daten

Zweck: Datenausgabe vom Computer zu einem externen Gerät (Bandgerät, Floppy Disk-Speicher, Drucker).

Bemerkung: Die Datenverbindung zu einem externen Gerät ist durch eine vorangegangene OPEN- (oder DOPEN-) Anweisung geöffnet worden. Die logische Dateinummer Id in einer PRINT#-Anweisung stellt die Beziehung zur zugehörigen OPEN-Anweisung her.
Die Daten in der Parameterliste werden durch Trennsymbole c$ getrennt.

1. PRINT#-AUSGABE ZUM BANDGERÄT

Jede Zahl oder Zeichenkette, die in eine Banddatei geschrieben wird, muß mit dem Symbol für Wagenrücklauf abgeschlossen werden. PRINT#-Anweisungen senden automatisch das Wagenrücklauf-Symbol mit Ausgabe des letzten Elements daten in der Parameterliste. Zwischen den Elementen daten der Parameterliste muß das Wagenrücklauf-Symbol in Form des Trennsymbols c$ = CHR$ (13) eingefügt werden. Die Kodierung CHR$ (13) für Wagenrücklauf wird entweder direkt in der Parameterliste verwendet oder stellvertretend durch eine Variable, der zuvor das Wagenrücklauf-Symbol CHR$ (13) zugewiesen wurde.

```
Anweisungen:    Bandaufzeichnung:
                    A$ [WR] B$ [WR] A [WR] B [WR]
                       ↑       ↑       ↑       ↑
                  DURCH  W$      W$      W$      AUTOMATISCH
40 W$=CHR$(13)
50 PRINT#1,A$;W$;B$;W$;A;W$;B
```

2. PRINT#-AUSGABE ZUM FLOPPY DISK-SPEICHER

Für das Trennsymbol c$ gelten die gleichen Regeln wie für die Datenausgabe auf ein Bandgerät. Eine Ausnahme gilt jedoch: Besteht ein Element daten der Parameterliste aus mehreren Zeichenketten, dann können die einzelnen Zeichenketten durch ein Komma-Symbol getrennt werden. In diesem Fall wird innerhalb eines Elements daten das Trennsymbol c$ = CHR$ (44) zur Darstellung eines Kommas verwendet. Beim anschließenden Lesen von der Diskette genügt eine einzelne INPUT#-Anweisung, um alle durch Komma getrennten Zeichenketten zwischen zwei Wagenrücklauf-Symbolen zu lesen.

```
20 W$=CHR$(13) : K$=CHR$(44)   DISKETTENAUFZEICHNUNG:
30 PRINT#1,A$,K$,B$,W$,C,W$,B
                                   A$,B$ [WR] A [WR] B [WR]
                                     ↑       ↑       ↑       ↑
                              DURCH  K$      W$      W$      AUTO
                                                             MATISCH
```

3. PRINT#-AUSGABE ZUM DRUCKER

Die Elemente daten in der Parameterliste einer PRINT #-Anweisung werden nur mit den Symbolen c$ = CHR$ (29) getrennt; die unter Format angegebenen Semikolons entfallen.
In BASIC < 3.0 beendet jede PRINT #-Anweisung automatisch ihre Ausgaben an den Drucker mit den Steuercodes für Wagenrücklauf (WR) und Zeilenvorschub (ZV).
In BASIC 4.0 schließt PRINT # bei logischen Dateinummern 1 . . . 127 mit einem WR, bei 128 . . . 255 mit einem WR + ZV. Erfordert ein Drucker am Ende jeder Druckzeile den Steuercode ZV, dann muß dieser als CHR$ (10) am Ende der Parameterliste von PRINT # angefügt werden (s. S. 6-92).

```
DIREKTER DIALOG                    PROGRAMMFORM

BANDGERAET:                        10 OPEN 1,1,2
OPEN 1,1,2 : PRINT#1,"HALLO"       20 PRINT#1,"HALLO"
```

```
DRUCKER:                                    10 OPEN 1,4
OPEN 1,4 : CMD 1 : PRINT#1,"HALLO"          20 PRINT#1,"HALLO"
FLOPPY DISK:                                10 OPEN 1,8,15
OPEN 1,8,15 : PRINT#1,"HALLO"               20 PRINT#1,"HALLO"
```

PRINT#-LD, "COPY" — BASIC < 3.0

Format: PRINT#ld, "COPY z: zieldatei = q:quelldatei [,q:quelldatei . . .]"
PRINT#ld, "Cz:zieldatei = q:quelldatei [q:quelldatei . . .]"

Zweck: Kopieren und/oder Verketten von Dateien innerhalb des gleichen Floppy Disk-Speichers. Vergleiche die Anweisungen COPY und CONCAT in BASIC 4.0.

Bemerkung: Mit der angegebenen Anweisung können bis zu vier Quelldateien verkettet und in eine zieldatei kopiert werden. Hierbei werden die Quelldateien nicht verändert. Die Quelldateien werden durch ihren Namen quelldatei und die Laufwerknummer q gekennzeichnet. Bei Angabe von mehr als einer Quelldatei werden die Dateien in der Ordnung verkettet, in der sie in der PRINT#-Anweisung erscheinen. Die jeweils neueingerichtete Zieldatei wird durch den Namen zieldatei und eine Laufwerknummer z gekennzeichnet.

Beispiel:

```
DIREKTER DIALOG

OPEN 1,8,15 : PRINT#1,"C1:DAT#1=C0:DAT#0"
OPEN 1,8,15 : PRINT#1,"C0:NEU=C1:ALT#1,C0:ALT#0"    *)

*)DATEI "ALT#0" WIRD AN DAS ENDE VON DATEI "ALT#1"
  ANGEFUEGT UND MIT DEM RESULTAT DIE NEUE DATEI "NEU"
  AUF DISKETTE 0 GEBILDET.
```

PRINT#-LD, "DUPLICATE" — BASIC < 3.0

Format: PRINT#ld, "DUPLICATE d = o"
PRINT#ld, "D d = o"

Zweck: Duplizieren einer Diskette. Vergleiche die Anweisung BACKUP in BASIC 4.0.

Bemerkung: Die Original-Diskette befindet sich in Laufwerk o, das Duplikat in Laufwerk d. Die Duplizierung umfaßt den Namen der Diskette, ihre Kennzahl und alle Dateien auf der Diskette.
Vor Duplizierung der Original-Diskette ist es ratsam, den Schreibschutz der Original-Diskette durch Überkleben des dafür vorgesehenen Schlitzes in der Schutzhülle zu aktivieren, um sie vor den Folgen einer versehentlichen Verwechslung von Original und Duplikat zu schützen.

Beispiel:

```
DIREKTER DIALOG                                  PROGRAMMFORM

OPEN 1,8,15 : PRINT#1,"DUPLICATE0=1"
OPEN 1,8,15 : PRINT#1,"D0=1"
```

PRINT#-LD, "INITIALIZE" — BASIC < 3.0

Format: PRINT#ld, "INITIALIZE [lfw]"
PRINT#ld, "I [lfw]"

Zweck: Intiieren einer neuen Diskette vor ihrer Verwendung zur Speicherung. Diese Operation ist nicht erforderlich in BASIC 4.0 und beim Arbeiten mit Disketten-Betriebssystemen der Versionen 2.x.

Bemerkung: Die Diskette in Laufwerk lfw wird initiiert. Bei fehlender Angabe der Laufwerknummer lfw werden beide Disketten im Floppy Disk-Speicher initiiert. Diese Initiierung ist beim Arbeiten mit BASIC < 3.0 und den Vorläufern des Disketten-Betriebssystems DOS 2.x erforderlich. Die Initiierung erfolgt automatisch in DOS 2.x.
Wird eine neue Diskette mit der nachfolgend beschriebenen Anweisung NEW erstmalig angelegt, dann erübrigt sich eine Initiierung.

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

OPEN 1,8,15 : PRINT#1,"INITIALIZE"
OPEN 1,8,15 : PRINT#1,"I"
```

PRINT#-LD, "NEW" — BASIC < 3.0

Format: PRINT#ld, "NEW lfw: diskname, vv"
PRINT#ld, "N lfw: diskname, vv"

Zweck: Formatieren einer neuen Diskette oder Wiederholung der Formatierung einer alten Diskette. Vergleiche die Anweisung HEADER in BASIC 4.0.

Bemerkung: Die Diskette in Laufwerk lfw wird angelegt, wobei ihr der Name diskname und die Kennzahl vv gegeben wird. Beim Anlegen einer Diskette werden die Sektoren (Blöcke) auf der Oberfläche der Diskette markiert, sowie das Inhaltsverzeichnis gespeicherter Dateien und das Verzeichnis verfügbarer Blöcke (BAM) angelegt. Der Name der Diskette diskname und ihre Kennzahl vv werden bei Darstellung des Inhaltsverzeichnisses auf dem Bildschirm als Kopfeintrag in Negativschrift dargestellt.
Beim Anlegen einer alten Diskette können Sie einen neuen Namen diskname vergeben, und hierbei die alte Kennzahl vv beibehalten oder ebenfalls ändern. Sie können jedoch lediglich die Kennzahl vv ändern.

Beispiel:

```
DIREKTER DIALOG

OPEN 1,8,15 : PRINT#1,"N0:NEU,02"          *)

OPEN 1,8,15 : PRINT#1,"N1"                 #)1.
OPEN 1,8,15 : PRINT#1,"N1:ALT#2"           #)2.
OPEN 1,8,15 : PRINT#1,"N1:ALT#2,02"        #)3.

 FABRIKNEUE DISKETTEN:
*)INITIIEREN EINER DISKETTE IN LAUFWERK 0 (N0)UNTER DEM
 DISKETTEN-NAMEN "NEU" UND DER KENNZAHL 02
 GEBRAUCHTE DISKETTEN:
#)INITIIEREN EINER DISKETTE IN LAUFWERK 1 (N1) UNTER
 1.BEIBEHALTEN NAMENS UND DER KENNZAHL  2.AENDERN DES
 ALTEN NAMENS IN ALT#2  3.AENDERN VON NAME UND KENNZAHL
```

PRINT#-LD, "RENAME" BASIC < 3.0

Format: PRINT#ld, "RENAME lfw: neuname = altname"
PRINT#ld, "R lfw: neuname = altname"

Zweck: Ändern des Namens einer Disketten-Datei. Vergleiche auch die Anweisung RENAME in BASIC 4.0.

Bemerkung: Der alte Dateiname altname einer Datei auf der Diskette in Laufwerk lfw wird geändert in den Dateinamen neuname.

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

OPEN 1,8,15 : PRINT#1,"R1:NEU=ALT"
```

PRINT#-LD, "SCRATCH" BASIC < 3.0

Format: PRINT#ld, "S lfw: dateiname [,lfw: dateiname]"

Zweck: Löschen einer oder mehrerer Dateien auf einer Diskette. Vergleiche auch die Anweisung SCRATCH in BASIC 4.0.

Bemerkung: Mit einer einzelnen PRINT#-Anweisung können eine oder mehrere Dateien mit den Namen dateiname auf einer oder beiden Disketten eines Floppy Disk-Speichers in den Laufwerken lfw gelöscht werden. Zum Löschen von Dateien mit ähnlich lautenden Namen können vorteilhaft die Symbole (*) und (?) verwendet werden. Das Sternsymbol (*) steht entweder stellvertretend für alle Dateinamen, so daß bei Eingabe des Dateinamens "*" sämtliche Dateien auf der Diskette gelöscht werden. Oder das Sternsymbol steht stellvertretend für den veränderlichen Teil von Dateinamen, deren Anfangszeichen gleich sind. Ein Beispiel: Bei Eingabe des Dateinamens "DAT*" werden sämtliche Dateien gelöscht, deren Name mit den Anfangsbuchstaben DAT beginnt. Das Fragezeichen (?) in der Angabe eines Dateinamens gibt an, an welcher Stelle die Namen zu löschender Dateien beliebige Zeichen enthalten können. Ein Beispiel: Durch Angabe des Dateinamens "D???N" wird jede Datei gelöscht, deren Name mit dem Buchstaben D beginnt und mit dem Buchstaben N endet, und zwischen diesen beiden Buchstaben drei beliebige Zeichen enthält.

Beispiel:

```
DIREKTER DIALOG

OPEN 1,8,15 : PRINT#1,"S0:PROGR#1"
OPEN 1,8,15 : PRINT#1,"S0:PR#1,1:PR#2" 10 OPEN 1,8,15
OPEN 1,8,15 : PRINT#1,"S0:PR#*,1:P???" .
OPEN 1,8,15 : PRINT#1,"S0:*"     500 PRINT#1,"S0:PROGR#1"
```

PRINT#-LD, "VALIDATE" BASIC < 3.0

Format: PRINT#ld, "VALIDATE [lfw]"
PRINT#ld, "V [lfw]

Zweck: Löschen unvorschriftsmäßig geschlossener Disketten-Dateien und Streichen des Namens dieser Dateien im Inhaltsverzeichnis. Vergleiche auch die Anweisung COLLECT in BASIC 4.0.

Bemerkung: Die Dateien auf der Diskette in Laufwerk lfw werden revidiert. Bei fehlender Laufwerknummer lfw werden die Dateien auf der zuletzt benutzten Diskette revidiert.
Beim Revidieren einer Diskette wird ein neues Block-Verfügbarkeitsverzeichnis für alle auf der Diskette vorhandenen Dateien angelegt. Alle unvorschriftsmäßig geschlossenen oder nicht geschlossenen Dateien werden von der Diskette gelöscht und der von ihnen belegte Speicherplatz wieder zugänglich gemacht.
Wenden Sie die Revision einer Diskette nicht auf Dateien mit wahlfreiem Zugriff (USR) an; in diesem Fall führt die Revision zur Löschung dieser Dateien.
Ereignet sich bei der Revision ein Lesefehler, dann wird die Revision aufgegeben und die Diskette in ihrem ursprünglichen Zustand belassen. Eine Diskette muß nach erfolgter Revision wieder initiiert werden.

Beispiel:
```
DIREKTER DIALOG                                PRPROGRAMMFORM

OPEN 1,8,15: PRINT#1,"V0": PRINT#1,"I0"
```

READ

Format: READ var[,var,. . .,var]

Zweck: Zuweisung von Werten einer DATA-Anweisung zu den Variablen var in der READ-Anweisung.

Bemerkung: READ kann statt mehrfacher Zuweisungen mit LET stehen. Zwischen den Werten in DATA und den Variablen var in READ muß Übereinstimmung in Datentyp und Anzahl bestehen. Bei geringerer Anzahl von Werten in DATA wird die Fehlermeldung ? OUT OF DATA ausgegeben.

Beispiel:
```
DIREKTER DIALOG          PROGRAMMFORM

                         10 DATA 1,2,3
                         .       ↑ ↑ ↑
                         80 READ A,B,C

                         10 DATA 123,ABC,"A,B,C",123.456
                         .       ↑   ↑     ↑       ↑
                         80 READ  A , A$,   B$  ,   B
```

RECORD

BASIC 4.0

Format: RECORD#ld, aufzeichnung#[,byte#]

Zweck: Positionieren eines Zeigers auf ein Byte in der angegebenen Aufzeichnung einer Relativen Datei, um einen nachfolgenden Zugriff auf die Relative Datei mit GET#, INPUT# oder PRINT# vorzubereiten.

Bemerkung: In der Relativen Datei, die unter der logischen Dateinummer ld geöffnet wurde, wird das Byte mit der Nummer byte# in der Aufzeichnung

mit der Nummer aufzeichnung# für den Beginn eines nachfolgenden Zugriffs markiert.
Wird mit der Anweisung RECORD der Zeiger auf das Ende der Datei positioniert, dann führt ein nachfolgender Zugriff mit der Anweisung PRINT# zur Erweiterung der Datei, um PRINT# das Schreiben einer weiteren Aufzeichnung zu ermöglichen; besteht der nachfolgende Zugriff aus einer INPUT#-Anweisung, dann werden mit INPUT# keine Daten gelesen und das Statuswort ST auf "Dateiende" gesetzt.
Die Zahl der Aufzeichnungen in einer Relativen Datei beschränkt aufzeichnung# auf die Zahlenwerte 1 bis 65535. Die Beschränkung der Zeichenzahl je Aufzeichnung schränkt den Wertebereich für byte# auf die Zahlen 1 bis 254 ein. Bei fehlender Angabe von byte# wird auf Byte#1 positioniert.

Beispiel:

```
20 REM OEFFNEN DER RELATIVEN DATEI "DATEI" MIT 20 BYTES
25 REM LAENGE/AUFZEICHNUNG:
30 DOPEN#1,"DATEI",L20,D1
40 REM BEZEICHNEN VON BYTE 6 IN AUFZEICHNUNG 20:
50 RECORD#1,20,6
60 REM LESEN DES BEZEICHNETEN BYTE UND ZUWEISEN ZU A$:
70 GET#1,A$:IF A$="" THEN 30
80 DCLOSE#1
90 STOP
```

REM

Format: REM bemerkung

Zweck: Einfügen erklärender Bemerkungen in ein Programm.

Bemerkung: Die Anweisung REM und eine dahinterstehende Bemerkung des Programmierers wird bei Programmauflistung wiedergegeben, im Programmablauf jedoch ignoriert. Eine REM-Bemerkung kann als Programmzeile mit eigener Zeilennummer oder an das Ende einer beliebigen Programmzeile in das Programm eingefügt werden.
Eine REM-Anweisung darf nicht an den Anfang einer Programmzeile gesetzt werden, auf der noch weitere BASIC-Anweisungen folgen, da alle Angaben nach REM nur als Kommentar behandelt werden. Verzweigungen zur Zeilennummer einer REM-Anweisung sind zulässig.

Beispiel:

```
DIREKTER DIALOG  PROGRAMMFORM

                 10 REM ** UNTERPROGRAMM XY **
                 .
                 50 IF DS<>0 GOTO 90 : REM STATUSABFRAGE
```

RENAME **BASIC 4.0**

Format: RENAME [lfw] "altname" TO "neuname" [ON Ugn]

Zweck: Ändern des Namens einer Disketten-Datei. Vergleiche auch die Anweisung PRINT#-LD, "RENAME".

Bemerkung: Die Datei mit dem Namen altname auf der Diskette in Laufwerk lfw erhält den neuen Namen neuname. Bei fehlender Angabe einer Lauf-

werknummer lfw wird Laufwerk 0 angenommen.
Ergeben sich bei der Neubenennung einer Datei Schwierigkeiten, dann versuchen Sie zunächst eine Revision der Diskette mit der Anweisung COLLECT oder PRINT#, "VALIDATE".

Vorsicht: Eine Datei muß vor Neubenennung geschlossen werden.

Beispiel:

DIREKTER DIALOG

```
RENAME "PROGR #1" TO "PROGR #2"
RENAME D1,"PROGR #1" TO "PROGR #2"
```

PROGRAMMFORM

RESTORE

Format: RESTORE

Zweck: Rücksetzen des Lesezeigers zum wiederholten Lesen der Daten in DATA-Anweisungen.

Bemerkung: Eine nach RESTORE folgende READ-Anweisung beginnt das Lesen von Daten mit dem ersten Datum in der ersten DATA-Anweisung des Programms.

Beispiel:

DIREKTER DIALOG

PROGRAMMFORM

```
10 READ A,B,C
20 RESTORE
30 READ X,Y,Z
.
90 DATA 10,30,50
```

RETURN

Format: RETURN

Zweck: Rücksprung aus einem Unterprogramm in das aufrufende Hauptprogramm.

Bemerkung: Eine RETURN-Anweisung am Ende eines Unterprogramms löst den Rücksprung zum nächsten Befehl im Hauptprogramm, der dem zuletzt ausgeführten GOSUB-Befehl folgt. Die Anweisung RETURN und die mit den gleichen Buchstaben gekennzeichnete Taste RETURN haben vollkommen verschiedene Funktionen.

Beispiel:

DIREKTER DIALOG

PROGRAMMFORM

```
100 GOSUB 2000
110 ..........
 .
2000 ......
 .
2100 RETURN
```

RUN

Format: RUN [zn]

Zweck: Auslösen des Ablaufs eines im Computer gespeicherten Programms.

Bemerkung: Die Anweisung RUN schließt alle offenen Dateien, initiiert alle Variablen auf den Wert 0 und setzt die Zeiger im Arbeitsspeicher des Computers zurück, ehe der Programmablauf ausgelöst wird. Vergleiche für die ersten Operationen von RUN die Anweisungen CLR und RESTORE.

Wenn RUN eine Zeilennummer zn angibt, dann werden die Funktionen von CLR und RESTORE ausgeführt, der Programmablauf beginnt aber mit der angegebenen Zeilennummer des Programms.

Die Anweisung RUN zn sollte nicht nach einer Programmunterbrechung verwendet werden; benutzen Sie für diesen Zweck CONT oder GOTO.
Die Anweisung RUN kann auch in Programmform verwendet werden; hierdurch wird der Programmablauf wiederholt, nachdem alle Variablen gelöscht und Zeiger rückgesetzt worden sind.

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

RUN
RUN 100     *)

*)INITIIEREN ALLER VARIABLEN;PROGRAMM IM COMPUTER AB
  ZEILE 100 ABLAUFEN LASSEN.
```

SAVE

BASIC < 3.0

Format SAVE ["programmname"] [,gn] [,sa] BANDGERÄTE

Zweck: Speichern eines Programms auf der Kassette eines Bandgerätes. Vergleiche auch die Anweisung DSAVE in BASIC 4.0.

Bemerkung: Die Anweisung SAVE schreibt ein augenblicklich im Computer gespeichertes Programm unter dem Namen programmname auf das Bandgerät mit der Gerätenummer gn, wobei eine von 0 verschiedene Sekundäradresse sa das Schreiben einer Dateiende-Marke auslöst, sobald das Programm gespeichert ist. Obwohl keiner der Parameter in einer SAVE-Anweisung erforderlich ist, sollten Programme wenigstens mit einem Namen versehen werden. Ein Programm kann auf einer Bandkassette entweder durch seinen Namen oder durch die Stelle, an der es gespeichert ist, wiedergefunden werden. Bei fehlendem Programmnamen bleibt nur die Kenntnis der Lage der Programmaufzeichnung auf dem Band, um das Programm wiederzufinden.

Die Anweisung SAVE findet ihre häufigste Anwendung in direktem Dialog, obwohl sie auch durch ein Programm ausgeführt werden kann.
Vergleiche auch die Beschreibungen in Kapitel 2.

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

SAVE
SAVE "PROGR #1"

A$="PROGR #1"
SAVE A$

SAVE "PROGR #1",2,2
```

Format: SAVE "lfw: programmname", 8 FLOPPY-DISK-SPEICHER

Zweck: Speichern eines augenblicklich im Computer befindlichen Programms auf der Diskette eines Floppy Disk-Speichers.

Bemerkung: Eine SAVE-Anweisung schreibt eine Kopie des augenblicklich im Computer gespeicherten Programms unter dem Programmnamen programmname auf die Diskette in Laufwerk lfw des angeschlossenen Floppy Disk-Speichers. In allen CBM-Computer-Systemen tragen Floppy Disk-Speicher die Gerätenummer 8. Der verwendete programmname muß neu sein. Besteht bereits eine Datei mit dem gleichen Namen auf der Diskette, dann wird ein Syntax-Fehler gemeldet. In BASIC 4.0 und mit dem Betriebssystem DOS 2.x kann jedoch ein auf der Diskette vorhandenes Programm gleichen Namens mit der Anweisung SAVE überschrieben werden, wenn der Laufwerknummer lfw das Symbol @ vorangesetzt ist.
Die Anweisung SAVE wird hauptsächlich in direktem Dialog verwendet.

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

SAVE "0:PROGR#1",8      *)
SAVE "@0:PROGR#1",8     +)

  DAS PROGRAMM IM COMPUTER AUF DISKETTE 0 SPEICHERN
*)UNTER DEM NAMEN "PROGR#1"
+)UNTER UEBRESCHREIBEN DES AUF DER DISKETTE VORHANDENEN
  PROGRAMMS "PROGR#1"
```

SCRATCH **BASIC 4.0**

Format: SCRATCH [Dlfw], "dateiname" [ON Ugn]

Zweck: Löschen einer Datei auf Diskette.

Bemerkung: Die Datei mit dem Namen dateiname auf der Diskette in Laufwerk lfw wird gelöscht. Bei fehlender Angabe einer Gerätenummer gn wird automatisch ein Floppy Disk-Speicher mit der Gerätenummer gn = 8 angenommen.
Bei Ausführung der SCRATCH-Anweisung in direktem Dialog erscheint die Rückfrage "ARE YOU SURE?" (Sind Sie sicher?) auf dem Bildschirm. Um die Löschung der Datei zu veranlassen, müssen Sie als Antwort YES oder Y eingeben und die Taste RETURN drücken.

Bei Ausführung einer SCRATCH-Anweisung in Programmform erscheint keine Rückfrage auf dem Bildschirm. Für kurzzeitig benötigte

Daten werden häufig von einem Programm Dateien angelegt, die vor Programmabschluß wieder gelöscht werden müssen, um einen erneuten Programmablauf zu ermöglichen; wird eine kurzzeitig benötigte Datei nicht gelöscht, dann löst ein erneuter Programmablauf die Fehlermeldung FILE EXISTS aus, da die Datei noch vom letzten Programmlauf auf der Diskette steht.
Dateien müssen vor dem Löschen geschlossen werden. Wenn Sie versuchen, eine offene Datei zu löschen, dann kann das zu komplizierten und fehlerhaften Operationen des CBM-Computers führen.
Im Betriebssystem DOS 2.x sollte eine Diskette in direktem Dialog mit der Anweisung COLLECT revidiert werden, ehe eine Datei auf der Diskette gelöscht wird.

Beispiel:

```
DIREKTER DIALOG                    PROGRAMMFORM

SCRATCH "PROGR#1"                  470 SCRATCH D0,"PROGR#1"
SCRATCH D0,"PROGR#1"
SCRATCH D1,"PROGR#1"
```

STOP

Format: STOP

Zweck: Beenden eines Programmablaufs und Rückkehr zu direktem Dialog.

Beispiel:

```
DIREKTER DIALOG                    PROGRAMMFORM

                                   480 STOP
```

VERIFY

Format: VERIFY ["programmname"][,gn] BANDGERÄTE

Zweck: Vergleich eines eben auf Diskette gespeicherten Programms mit dem noch im Computer vorhandenen Original und Meldung von Abweichungen.

Bemerkung: Das eben unter dem Namen programmname auf dem Bandgerät mit der Gerätenummer gn abgespeicherte Programm wird mit dem noch im Speicher des Computers vorhandenen Original verglichen. Bei fehlender Gerätenummer gn wird das Bandgerät # 1 gewählt. Bei fehlendem Programmnamen wird das nächste, auf der Kassette angetroffene Programm mit dem Original verglichen.
Ein Vergleich sollte immer unmittelbar nach dem Abspeichern eines Programms erfolgen. Die Anweisung VERIFY wird fast ausschließlich in direktem Dialog verwendet.

Beispiel:

```
DIREKTER DIALOG                    PROGRAMMFORM

VERIFY
VERIFY "PROGR#1"

A$="PROGR#1"
VERIFY A$
```

Format: VERIFY "lfw: programmname", 8

Zweck: Vergleich eines eben auf Diskette gespeicherten Programms mit dem noch im Speicher des Computers vorhandenen Original und Meldung von Abweichungen.

Bemerkung: Das eben auf die Diskette in Laufwerk lfw unter dem Namen programmname abgespeicherte Programm wird mit dem im Speicher des Computers noch vorhandenen Original verglichen.
Zur Überprüfung eines gerade eben auf Diskette abgespeicherten Programms kann die Anweisung VERIFY auch in folgender abgekürzter Form verwendet werden:

```
VERIFY "*",8
```

Beispiel:

```
DIREKTER DIALOG                                    PROGRAMMFORM

VERIFY"*",8
VERIFY "0:PROGR#1",8

A$="PROGR#1"
VERIFY A$
```

WAIT

Format: WAIT speicheradresse, maske [, exor]

mit:

maske	Maske von 1 Byte Länge
exor	Maske von 1 Byte Länge

Zweck: Aussetzen des Programmablaufs, bis eine angegebene Speicherstelle einen ebenfalls angegebenen Inhalt angenommen hat.

Bemerkung: Die Anweisung WAIT führt folgende Schritte aus:

1. Der Speicherinhalt unter der angegebenen Speicheradresse wird gelesen.
2. Der gelesene Speicherinhalt und das Bit-Muster exor werden in einer EXCLUSIVE-OR-Operation logisch verknüpft. Bei fehlender Maske exor wird dieser Schritt übersprungen.
3. Das in Schritt 2 erhaltene Bit-Muster wird mit dem Bit-Muster maske in einer logischen AND-Operation verknüpft. Diese Verknüpfung findet zwischen dem gelesenen Speicherinhalt und dem Bit-Muster maske statt, wenn eine Angabe für exor fehlt.
4. Ist das Ergebnis von Schritt 3 ein Bit-Muster 0, dann erfolgt der Rücksprung zu Schritt 1. Hierdurch ergibt sich eine Warteschleife, die erst in Schritt 5 beendet wird.
5. Liefert Schritt 3 nicht als Ergebnis das Bit-Muster 0, dann setzt der Programmablauf mit der nach WAIT folgenden Anweisung fort.

Eine WAIT-Anweisung kann nicht mit der Taste STOP unterbrochen werden.

Funktionen in CBM BASIC werden nachfolgend in alphabetischer Reihenfolge beschrieben. Einige wenige Funktionen sind nur auf Computern der Serie CBM 8001 verfügbar; diese Funktionen werden in einem nachfolgenden Abschnitt beschrieben.

ABS

Format: ABS (zahl)

Zweck: Bilden des Absolutwerts einer Zahl. Statt einer Zahl kann auch ein Ausdruck angegeben werden, dessen Ausführung eine Zahl als Resultat hat.

Beispiel:

```
DIREKTER DIALOG

?ABS(10)
 10
?ABS(-10)
 10
```

```
PROGRAMMFORM

20 A=10:B=-10
30 PRINT ABS(A),ABS(B)
```

ASC

Format: ASC (zeichen $)

Zweck: Verschlüsseln eines angegebenen Zeichens im ASCII-Kode.

Bemerkung: Besteht das angegebene Zeichen zeichen$ aus mehr als einem Zeichen, dann liefert die Anweisung ASC den ASCII-Kode für das erste Zeichen in der Zeichenkette. Das Ergebnis ist eine Zahl, die in arithmetischen Operationen weiterverwendet werden kann. (Siehe Anhang A für Verschlüsselungen im ASCII-Kode).

Beispiel:

```
DIREKTER DIALOG

?ASC("A")
 65
```

```
PROGRAMMFORM

20 POKE 32768,ASC("↑")-128

BILDSCHIRMSPEICHER MIT ADRESSE 32768
WIRD MIT ASCII-KODE FUER "↑" GELADEN
```

ATN

Format: ATN (zahl)

Zweck: Die Funktion ATN liefert den Wert des Arkustangens des angegebener Arguments zahl.

Bemerkung: Dimension des Ergebnisses ist Radian; Wertebereich des Ergebnisses ist $-\pi/2$ bis $+\pi/2$. Das Argument zahl kann eine Zahl oder ein Ausdruck

sein; die Berechnung von ATN erfolgt jedoch immer für Gleitkommazahlen.

Beispiel:

```
DIREKTER DIALOG                PROGRAMMFORM

?ATN(1)                        20 A=ATN(2*X+Y)
 .785398163   [RADIAN]         30 PRINT A

                               [A]=RADIAN
```

CHR$ ( )

Format: CHR$ (dezimalzahl)

Zweck: Verschlüsselte Darstellung von Zeichen und Tastenfunktionen, die ohne Verschlüsselung nicht in einem Programm verwendet werden könnten.

Bemerkung: Die Funktion CHR$ ( ) dient zur Verschlüsselung von Zeichen wie (, ; ") und Tastenfunktionen wie (RETURN, HOME, CLEAR). Die Verschlüsselung von Zeichen und Funktionen erfolgt im ASCII-Kode, dessen Zahlenwert zahl als Argument in CHR$ erscheint. Siehe Anhang A, Tabelle A-4.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM
?CHR$(65)                20 IF C$=CHR$(13) GOTO 100     *)
A                        50 PRINT CHR$(34);"HORROR";CHR$(34)

?CHR$(34);"HORROR";CHR$(34)
"HORROR"

  VERGLEICHE MIT:
?"HORROR"                *)CHR$(13)=KODE FUER DIE TASTE "RETURN"
HORROR
```

COS

Format: COS (zahl)

Zweck: Trigonometrischer Kosinus.

Bemerkung: Die Funktion COS liefert den Kosinus des in Radian angegebenen Arguments zahl.

Beispiel:

```
DIREKTER DIALOG              PROGRAMMFORM
?COS(45/180*3.1415926)       20 C=COS(XR)
 .707106791

KOSINUS VON 45 GRAD          XR[RADIAN]
```

DS

BASIC 4.0

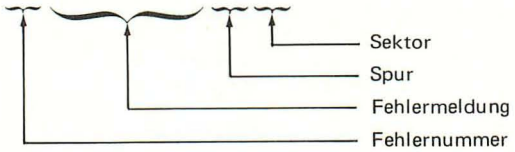
Zweck: Die Variable DS enthält Information über Fehler, die sich beim letzten Zugriff auf eine Diskette ereignet haben. Fehler werden mit Ganzzahlen verschlüsselt, die in Tabelle 6-2 zusammengestellt sind und in einem Programm abgefragt werden können.

Beispiel:

```
20 IF DS<>0 THEN PRINT "FEHLERMELDUNG" : STOP
```

Format:

```
NN FEHLERMELDUNG TT SS
```



Zweck: Die Variable DS$ bringt Information über einen Fehler, der sich beim Zugriff auf eine Diskette ereignet hat, in Form einer Fehlermeldung auf dem Bildschirm zur Darstellung. Anhang B enthält eine Zusammenstellung derartiger Fehlermeldungen.
Hat die Variable DS den Wert 1, dann wurde eine Datei gelöscht; jeder andere Wert von DS, der kleiner als 20 ist, stellt keinen Fehler beim Zugriff auf Disketten dar.

Beispiel:

```
20 IF DS>20 THEN PRINT DS$:STOP
```

EXP

Format: EXP (zahl)

Zweck: Exponentialfunktion

Bemerkung: Die Funktion EXP exponentiert das Argument zahl in der Form e^{zahl}. Der Wert von e ist e = 2.71828183.

Beispiel:

```
DIREKTER DIALOG                                  PROGRAMMFORM

?EXP(0)                                          20 E=EXP(X+Y/Z)
 1                                               30 PRINT E
?EXP(1)
 2.71828183
?EXP(-88.0296919)     *)
 5.87747176E-39
?EXP(88.029691)       +)
 1.70141025E+38

    *)KLEINSTES  +)GROESSTES VERARBEITETES ARGUMENT
```

FRE

Format: FRE (zahl)

Zweck: Angabe des unbelegten Speicherplatzes im Computer.

Bemerkung: Die Funktion FRE verwendet ein Scheinargument. Es kann eine Zahl oder eine Zeichenkette sein. FRE meldet die im Computer nicht belegten Bytes des Speichers; die Funktion wird gewöhnlich in direktem Dialog zusammen mit einer PRINT-Anweisung verwendet.

Beispiel:

```
DIREKTER DIALOG                                  PROGRAMMFORM

?FRE(1)        ANZEIGE DES VERFUEGBAREN,
 31436           FREIEN SPEICHERPLATZES
```

INT

Format: INT (zahl)

Zweck: Rückmeldung des ganzzahligen Anteils einer gegebenen Zahl.

Bemerkung: Bei positiven Zahlen bewirkt die Funktion INT das Abschneiden eines etwa vorhandenen Dezimal-Anteils ohne Rundung. Bei negativen Zahlen bewirkt die Funktion INT das Abschneiden des Dezimal-Anteils der Zahl und die Addition einer 1. Beachten Sie, daß die Funktion INT nicht die Umwandlung einer Gleitkommazahl (5 Bytes) in eine Ganzzahl (2 Bytes) bewirkt.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?INT(1.5)                20 A=-0.1
 1                       30 PRINT INT(A)
```

Vorsicht: Da Gleitkommazahlen nur Annäherungen an reale Zahlen darstellen, kann die Funktion INT möglicherweise nicht den erwarteten ganzzahligen Anteil eines Arguments liefern. Ein Beispiel: Das Argument zahl = 3.89999999 liefert als Ergebnis die Zahl 3, und nicht wie erwartet die Zahl 4:

```
?INT(3.89999999)
3
```

LEFT$

Format: LEFT$ (zeichenkette, anzahl)

Zweck: Rückmelden der linksstehenden Zeichen einer Zeichenkette.

Bemerkung: Der Parameter byte gibt an, wieviele der linksstehenden Zeichen zurückzumelden sind.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?LEFT$("ABCDEF",2)       20 A$=LEFT$("ABCDEF",2)
AB                       30 PRINT A$
```

LEN

Format: LEN (zeichenkette)

Zweck: Rückmelden der Länge einer Zeichenkette.

Bemerkung: Die Funktion LEN liefert eine Zahl, die das Ergebnis der Zählung aller Zeichen in einer Zeichenkette zeichenkette ist.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?LEN("ABCDEF")           20 N=LEN("ABCDEF")
 6                       30 PRINT N
```

LOG

Format: LOG (zahl)

Zweck: Natürlicher Logarithmus.

Bemerkung: Das Argument zahl muß größer als 0 sein. Die Funktion LOG liefert den Logarithmus zur Basis e = 2.71828183.
Bei negativem Argument oder beim Null-Argument erscheint die Fehlermeldung ILLEGAL QUANTITY ERROR.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?LOG(1)                  20 A=LOG(X)/LOG(10)
 0
?LOG(10)                     (BERECHNUNG DES LOGARITHMUS VON X
 2.30258509                   ZUR BASIS 10)
```

MID$

Format: **MID$ (zeichenkette, n [,l])**

Zweck: Bildung beliebiger Ausschnitte aus einer gegebenen Zeichenkette.

Bemerkung: Die Funktion MID$ bildet aus einer gegebenen Zeichenkette einen Ausschnitt, der, von links gezählt, mit dem n-ten Zeichen der Zeichenkette beginnt und eine Länge von l Zeichen besitzt. Für das erste Zeichen gilt n = 1. Bei fehlender Längenangabe l besteht der Ausschnitt aus allen Zeichen ab Zeichen n bis zum Ende der Zeichenkette. Entsprechend der größten zulässigen Länge einer Zeichenkette sind n, l auf einen Wertebereich von 0 bis 255 beschränkt.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?MID$("ABCDEF",2,1)      20 A$=MID$("ABCDEF",2,1)
B
?MID$("ABCDEF",3,2)
CD
?MID$("ABCDEF",3)
CDEF
```

PEEK

Format: **PEEK (speicheradresse)**

Zweck: Lesen des Speicherinhalts eines beliebigen Speicherplatzes im CBM-Computer.

Bemerkung: Mit der Funktion PEEK kann der Speicherinhalt jedes Speicherplatzes mit der Adresse speicheradresse gelesen werden; ausgenommen hiervon sind Speicherbereiche, in denen sich urheberrechtlich geschützte Systemprogramme von CBM-Computern befinden. Beim Lesen dieser Bereiche liefert PEEK den Wert 0. Die Adressen speicheradresse müssen im Bereich 0 bis 65.535 liegen. Speicherinhalte werden von PEEK in Form von Dezimalzahlen im Bereich 0 bis 255 zurückgemeldet. Siehe Anhang A, Tabelle A-4 für die Interpretation dieser Dezimalzahlen.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?PEEK(32768)             20 PRINT PEEK(32768)
 4
```

POS

Format: POS (z)

Zweck: Melden der augenblicklichen Spaltenposition des Kursors.

Bemerkung: Das Argument z in POS ist ein Scheinargument, dem jeder Wert gegeben werden kann.
Die Funktion POS liefert die augenblickliche Kursor-Position in Form eines Zahlenwerts zwischen 0 und 39 für einen 40spaltigen Bildschirm (0 und 79 für einen 80spaltigen Bildschirm).

Beachten Sie folgendes: Auch bei einem Computer mit 40spaltigem Bildschirm sind Programmzeilen mit 80 Zeichen möglich. Befindet sich die Programmlogik in der zweiten Hälfte einer solchen Programmzeile, dann liefert die Funktion POS einen Zahlenwert zwischen 40 und 79, selbst wenn der Computer nur einen 40spaltigen Bildschirm besitzt.
Bei fehlender Kursor-Darstellung liefert POS die augenblickliche Schreibstellung auf einer Programmzeile oder am Ende einer Zeichenkette. Da Zeichenketten durch Verkettung aus bis zu 255 Zeichen bestehen können, kann die Funktion POS in diesem Fall einen Zahlenwert zwischen 0 und 255 melden.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?POS(1)                  20 IF POS(1)>60 THEN PRINT CHR$(13)
 0

?"ABCABC";POS(1)
ABCABC 6
```

RIGHT$

Format: RIGHT$ (zeichenkette, anzahl)

Zweck: Rückmelden einer angegebenen Anzahl rechtsstehender Zeichen in einer Zeichenkette.

Bemerkung: Die Funktion RIGHT$ liefert soviele der rechtsstehenden Zeichen einer gegebenen Zeichenkette, wie die Anzahl anzahl angibt. Für anzahl = 0 liefert RIGHT$ eine leere Zeichenkette. Für anzahl = LEN (zeichenkette) meldet RIGHT$ die angegebene Zeichenkette unverändert zurück.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?RIGHT$("ABCDEF",2)      20 R$=RIGHT$("ABCDEF",2)
EF                       30 PRINT R$
```

RND

Format: RND (–zahl) Zufallsfolge "zahl" wählen
RND (zahl) Zufallszahl aus der Zufallsfolge "zahl" ziehen

Zweck: Bilden von Zufallszahlen im Bereich 0 bis 1.

Bemerkung: Die Funktion RND bildet unzählig viele verschiedene Zufallsfolgen. Nach Wahl einer der Zufallsfolgen durch Angabe eines Arguments – zahl wird bei jedmaliger Ausführung der Funktion RND (zahl) eine neue Zufallszahl aus der gewählten Zufallsfolge gezogen.

Beispiel:

```
DIREKTER DIALOG                          PROGRAMMFORM

A=RND(-10) : ?A      10 A=RND(-10)
 3.73729563E-08      20 FOR I=1 TO 5
                     30 PRINT RND(10)
                     40 NEXT
FOR I=1 TO 5:PRINT RND(10):NEXT
 .484587721
 .648753088
 .828494246
 .847033357
```

Für das Argument zahl = 0 liefert RND Zufallszahlen mit Hilfe des Zeittaktes TI des Computer-eigenen Zeittakt-Gebers. Hiermit ist die zufällige Auswahl einer der Zufallsfolgen in folgender Form möglich:

```
RND(-RND(0))
```

Siehe Kapitel 5 für eine vollständige Beschreibung der Möglichkeiten von RND.

SGN

Format: SGN (zahl)

Zweck: Feststellen, ob eine Zahl positiv, negativ oder 0 ist.

Bemerkung: Die Funktion SGN liefert den Wert +1, wenn die im Argument angegebene Zahl positiv ist; SGN liefert den Wert 0, wenn die angegebene Zahl 0 ist und den Wert –1, wenn die angegebene Zahl negativ ist.

Beispiel:

```
DIREKTER DIALOG     PROGRAMMFORM

?SGN(-6)            20 IF SGN(X)>0 THEN PRINT "POSITIVE ZAHL"
-1
?SGN(0)
 0
?SGN(6)
 1
```

SIN

Format: SIN (zahl)

Zweck: Trigonometrische Sinusfunktion.

Bemerkung: Die Funktion SIN liefert den Sinus zum angegebenen Argument zahl in Form einer Gleitkommazahl. Das Argument zahl hat die Dimension Radian. Das Argument zahl kann auch ein Ausdruck sein, der zunächst ausgewertet wird.

Beispiel:

```
DIREKTER DIALOG                 PROGRAMMFORM

?SIN(45/180*3.1415926)          20 S=SIN(XR)
 .707106772

SINUS VON 45 GRAD               XR[RADIAN]
```

SPC

Format: `PRINT...SPC(x)...`

Zweck: Rechtsverschiebung des Kursors um eine angegebene Anzahl von Spalten.

Bemerkung: Die Funktion SPC wird nur im Zusammenhang mit PRINT-Anweisungen verwendet; das Argument x gibt an, um wieviele Spalten der Kursor von seiner augenblicklichen Stellung nach rechts verschoben werden soll. Hierbei vom Kursor überfahrener Text wird nicht verändert. Vergleich zwischen den Funktionen SPC und TAB: SPC bewirkt eine relative Kursorverschiebung, ausgehend von der augenblicklichen Kursorstellung, TAB eine absolute Kursorverschiebung, ausgehend von der ersten Spaltenposition einer Bildschirmzeile.

```
DIREKTER DIALOG
PRINT"JAHR"SPC(5)"1982"SPC(5)"1983"
JAHR      1982      1983

PROGRAMMFORM
50 PRINT"JAHR"SPC(5)"1982"SPC(5)"1983"
```

SQR

Format: `SQR (zahl)`

Zweck: Bilden der Quadratwurzel einer positiven Zahl.

Beispiel:

```
DIREKTER DIALOG                 PROGRAMMFORM

?SQR(25)                        20 A=SQR(4.84)
 5                              30 PRINT A
?SQR(25.5)
 5.04975247
```

ST

Format: `ST`

Zweck: Informationen über den Status eines externen Geräts und einer externen Datei nach einem eben abgeschlossenen Zugriff.

Bemerkung: Die Funktion ST liefert Informationen über den Verlauf eines eben abgeschlossenen Zugriffs auf ein externes Gerät (z.B. Bandgerät,

Floppy Disk-Speicher) und eine externe Datei in Form eines 8-Bit-Musters, zu dessen Abfrage ST über die Wahrheitsfunktion AND mit dem gesuchten Bit-Muster verglichen wird. Ebenso kann ST durch Vergleich mit dem Zahlenwert 0 für eine allgemeine Abfrage auf das Vorliegen einer Statusmeldung verwendet werden.
Siehe Kapitel 6 für Beispiele von Statusabfragen in Programmen und Tabelle 6-2 für die Verschlüsselung von Statusmeldungen in Bit-Mustern.

Beispiel:

```
DIREKTER DIALOG    PROGRAMMFORM

                   20 IF ST<>0 THEN GOTO 500
                   50 IF ST=4 THEN ?"?!! KURZER BLOCK"
```

STR$

Format: STR$(argn)

Zweck: Darstellung eines numerischen Arguments als Zeichenkette.

Bemerkung: Die Funktion STR$ wandelt den Datentyp eines gegebenen numerischen Arguments argn in den Datentyp Zeichenkette um.

Beispiel:

```
DIREKTER DIALOG    PROGRAMMFORM

?STR$(14.56)       20 ON LEN(STR$(N)) GOSUB 100,200,300
 14.56             .
                   30 INPUT "ALTER:";A
                   40 A$=STR$(A)
```

SYS

Format: SYS (speicheradresse)

Zweck: Sprung in ein Maschinenprogramm.

Bemerkung: Die Systemfunktion SYS ermöglicht den Sprung in ein Maschinenprogramm, dessen erste Programmzeile in der angegebenen Adresse speicheradresse steht. Der Wert von speicheradresse muß im Bereich 0 bis 65.535 liegen. Siehe Kapitel 7.

TAB

Format: PRINT . . . TAB(s) . . .

Zweck: Rechtsbewegung des Kursors zu einer angegebenen Spaltenposition.

Bemerkung: Die Funktion TAB bewegt den Kursor auf die Spaltenposition s + 1, wenn s das Argument in TAB war. Die Zählung der Spaltenpositionen auf dem Bildschirm beginnt mit Spalte 0. Befindet sich der Kursor bereits hinter der angegebenen Spaltenposition s + 1, dann erfolgt eine Positionierung des Kursors auf der nächsten Bildschirmzeile. Die

Funktion TAB kann nur im Zusammenhang mit PRINT-Anweisungen verwendet werden. Vergleiche auch die Funktion SPC.

Beispiel:

```
DIREKTER DIALOG

?TAB(0);"↑"
↑
?TAB(3);"↑"
   ↑
```

```
PROGRAMMFORM

20 PRINT "NAME"TAB(15)"BETRAG"

RUN 20
NAME            BETRAG
```

TAB-Funktion und TAB-Taste

Einige CBM-Computer verfügen über eine Tabuliertaste TAB, die innerhalb der Parameterliste einer PRINT-Anweisung benutzt werden kann, um Tabellenpositionen zu setzen, zu löschen oder den Kursor zur nächsten Tabellenspalte zu bewegen.

Tabellenpositionen werden gesetzt und gelöscht durch Bedienen der Tasten SHIFT und TAB oder mit der Funktion CHR$ (9). Setzen und Löschen einer Tabellenposition werden durch die gleichen Dateneingaben bewirkt: D.h. die wiederholte Ausführung einer SHIFT- und TAB-Tasteneingabe an der gleichen Bildschirmstelle löscht eine zuvor dort gesetzte Tabellenposition.

Die Tabulierfunktion kann in direktem Dialog oder in Programmform verwendet werden. In Programmform werden PRINT-Anweisungen verwendet, die den Kursor zu den gewünschten Spaltenpositionen bewegen und durch Tasteneingabe von SHIFT und TAB die Spaltenposition markieren. Nach Eingabe von Wagenrücklauf (RETURN) werden die angegebenen Spaltenpositionen wirksam.

In nachfolgenden PRINT-Anweisungen bewirkt die Eingabe der Taste TAB oder der Funktion CHR$ (137), daß der Kursor zur nächsten Tabelierspalte bewegt wird, ehe die nächste, in PRINT enthaltene Variable zur Darstellung gebracht wird.

```
DIREKTER DIALOG

"■     ■     ■     ■"
?1,2,3,4
1     2     3     4
```

```
PROGRAMMFORM

20 PRINT "■     ■     ■     ■"
30 PRINT "1■2■3■4"

RUN20-30
 1    2    3    4
```

TAN

Format: TAN (zahl)

Zweck: Trigonometrische Tangensfunktion.

Bemerkung: Die Funktion TAN liefert den Tangens eines in Radian angegebenen Arguments zahl.

Beispiel:

```
DIREKTER DIALOG

?TAN(45/180*3.14159265)
 .999999998

TANGENS VON 45 GRAD
```

```
PROGRAMMFORM

20 T=TAN(XR)

XR[RADIAN]
```

TI, TI$

Format:

TI	Zahl der Zeittakte (numerische Variable)
TI$	Tageszeit (Zeichenketten-Variable)

Zweck: Lesen einer Computer-internen Uhr.

Bemerkung: Die Variable TI ermöglicht, den Computer-internen Zeittakt-Geber zu lesen. Die Zeittakte bilden ein Zeitraster von 1/60stel Sekunde.
Die Variable TI$ stellt die Umwandlung der Zeittakte TI in die Darstellungsweise der Tageszeit dar.
Kapitel 5 gibt Beschreibungen für das Arbeiten mit TI und TI$.

Beispiel:

```
DIREKTER DIALOG

?TI
 286417
?TI$
011944
```

```
PROGRAMMFORM

20 X=RND(-TI)

   ZUFALLSFOLGEN:SIEHE KAPITEL 5
```

USR

Format: USR(arg)

Zweck: Sprung in ein Maschinen-Unterprogramm des Benutzers.

Bemerkung: Die Funktion USR übergibt einen Parameter arg an ein Benutzer-Unterprogramm in Maschinensprache, dessen Adresse in den Speichern mit den Speicheradressen 1 und 2 steht. Mit der Funktion USR kann auch ein Parameter aus dem Benutzer-Unterprogramm zurückgemeldet werden.
Siehe Kapitel 7 für weitere Beschreibungen der Funktion USR.

VAL

Format: VAL (zeichenkette)

Zweck: Rückmelden eines numerischen Anteils in einer Zeichenkette.

Bemerkung: Die Funktion VAL prüft die ersten Zeichen einer Zeichenkette auf das Vorhandensein eines der Zeichen +, –, $, 0, 1, 2, . . . 9. Ist keines dieser Zeichen vorhanden, dann liefert VAL den Wert 0. Trifft VAL auf die angegebenen Zeichen am Anfang einer Zeichenkette, dann meldet VAL diese Zeichen in numerischem Datenformat.

Beispiel:

```
DIREKTER DIALOG

?VAL("123ABC")
 123
?VAL("ABC123")
 0
```

```
PROGRAMMFORM

20 READ NAME$,ALTER$
30 IF VAL(ALTER$)<12 THEN PRINT "KIND"
```

CBM-Computer der Serie CBM 8001 bieten folgende Sonderfunktionen:

BELL

CHR$ (7)

Format: **oder:** Taste ESC + Taste RVS + Taste "g"

Zweck: Programmierbarer Signalton.

Bemerkung: Durch Aufnahme der Funktion CHR$ (7) oder entsprechender Tasteneingaben in die Parameterliste einer PRINT-Anweisung kann ein Signalton programmiert werden. Der Signalton ist auch bei Inbetriebnahme des Computers oder bei Überschreiten der 75. Spalte auf dem Bildschirm hörbar. Wurde das Bildschirmfenster durch besondere Funktionen eingeengt, dann ist der Signalton hörbar, sobald der Kursor fünf Spalten vor der rechten Begrenzung des Bildschirmfensters steht.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

PRINT CHR$(7)            100 PRINT CHR$(7)
```

LÖSCHEN EINER BILDSCHIRMZEILE — BASIC 4.0

CHR$ (21)

Format: **oder:** Taste ESC + Taste RVS + Taste "u"

Zweck: Löschen einer Bildschirmzeile und anschließende Verschiebung des darunterstehenden Textes um eine Zeile nach oben.

Bemerkung: Zum Löschen einer Bildschirmzeile wird die Funktion CHR$ (21) oder die angegebenen Tasteneingaben in die Parameterliste einer PRINT-Anweisung aufgenommen; die Bildschirmzeile, auf der der Kursor augenblicklich steht, wird gelöscht. Die Löschung einer Bildschirmzeile bewirkt nicht ihre Löschung im Speicher des Computers. Diese Funktion sollte daher nur zur Textedition auf dem Bildschirm und nicht zu Speicheränderungen verwendet werden.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?"■■■■■■■■               20 PRINT "■■■■■■■■
?"■■■■"CHR$(21)          20 PRINT "■■■■"CHR$(21)

"■"=TASTE HOME ; "■"=TASTE KURSOR NACH UNTEN
IN BEIDEN BEISPIELEN WIRD DIE 4.BILDSCHIRMZEILE GELOESCHT
```

LÖSCHEN DES ZEILENANFANGS

CHR$ (150)

Format: **oder:** Taste ESC + Taste RVS + Taste "V")

Zweck: Löschen einer Bildschirmzeile von ihrem Anfang bis zur augenblicklichen Kursorstellung.

Bemerkung: Zum Löschen des Zeilenanfangs einer Bildschirmzeile wird die Funktion CHR$ (150) oder entsprechende Tasteneingaben in die Parameterliste einer PRINT-Anweisung aufgenommen. Die Löschung bezieht sich auf diejenige Bildschirmzeile, auf der sich der Kursor augenblicklich befindet. Die Löschung betrifft nur den Bildschirm und läßt Speicherinhalte unverändert.

Beispiel:

```
DIREKTER DIALOG    PROGRAMMFORM

                   20 PRINT  TAB(20);CHR$(150)

                   LOESCHEN DER ERSTEN 20 ZEICHEN AUF
                   DER KURSOR-ZEILE
```

LÖSCHEN DES ZEILENENDES

Format: CHR$ (22)

oder: Taste ESC + Taste RVS + Taste "v"

Zweck: Löschen einer Bildschirmzeile von der augenblicklichen Kursorstellung bis zum Ende der Zeile.

Bemerkung: Zum Löschen eines Zeilenendes wird die Funktion CHR$ (22) oder entsprechende Tasteneingaben in die Parameterliste einer PRINT-Anweisung aufgenommen. Die Löschung betrifft diejenige Bildschirmzeile, auf der sich der Kursor augenblicklich befindet. Löschungen auf einer Bildschirmzeile lassen den Speicherinhalt unverändert.

Beispiel:

```
DIREKTER DIALOG    PROGRAMMFORM

                   20 PRINT  TAB(20);CHR$(22)

                   LOESCHEN DER LETZTEN ZEICHEN AUF DER
                   KURSOR-ZEILE,BEGINNEND MIT ZEICHEN 20
```

GRAPHISCHE SYMBOLE

Format: CHR$ (142)

oder: Taste ESC + Taste RVS + Taste "N"

Zweck: Umschalten zu einem Zeichensatz für graphische Symbole.

Bemerkung: Zum Umschalten des Zeichensatzes auf graphische Symbole wird die Funktion CHR$ (142) oder entsprechende Tasteneingaben in die Parameterliste einer PRINT-Anweisung aufgenommen. Die Rückkehr zum Zeichensatz für Texte leistet die Funktion CHR$ (14), die weiter unten beschrieben ist.

Beispiel:

```
DIREKTER DIALOG    PROGRAMMFORM

PRINT CHR$(142)    100 PRINT CHR$(142)
```

EINFÜGEN EINER BILDSCHIRMZEILE

Format:

CHR$ (149)

oder:

Taste ESC + Taste RVS + Taste "m"

Zweck: Einfügen einer leeren Bildschirmzeile an der augenblicklichen Kursor-Position.

Bemerkung: Zum Einfügen einer leeren Bildschirmzeile wird die Funktion CHR$ (149) oder entsprechende Tasteneingaben in die Parameterliste einer PRINT-Anweisung aufgenommen. Textdarstellungen unterhalb der eingefügten Bildschirmzeile werden um eine Zeile nach unten bewegt; die unterste Bildschirmzeile verschwindet hierbei nach unten auf dem Bildschirm:
Eine Zeileneinfügung verändert lediglich das Schirmbild, nicht jedoch die Speicherinhalte im Speicher des Computers; sie sollte daher nur für Bildschirm-Editionen verwendet werden.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

?"■■■■■"                 20 PRINT "■■■■■■
?"■■■■"CHR$(149)         20 PRINT "■■■■"CHR$(149)

"■"=TASTE HOME ; "■"=TASTE KURSOR NACH UNTEN
IN BEIDEN BEISPIELEN WIRD AUF BILDSCHIRMZEILE 4 EINE
ZEILE EINGEFUEGT
```

VERSCHIEBUNG VON BILDSCHIRMTEXT

Format:

	CHR$ (25)	Textverschiebung
oder:	Taste ESC + Taste RVS + Taste "q"	um 1 Zeile nach oben
	CHR$ (153)	
oder:	Taste ESC + Taste RVS + Taste "Q"	um 1 Zeile nach unten

Zweck: Textverschiebung auf dem Bildschirm um eine Zeile nach oben bzw. um eine Zeile nach unten.

Bemerkung: Die angegebenen Funktionen CHR$ (25) und CHR$ (153) oder ihnen entsprechende Tasteneingaben werden in der Parameterliste einer PRINT-Anweisung angewendet. Bei Textverschiebung nach oben wird von unten eine leere Bildschirmzeile eingefügt; eine vorhandene Kopfzeile auf dem Bildschirm verschwindet nach oben aus dem Bildschirm. Bei Textverschiebung nach unten wird eine leere Bildschirmzeile von oben eingefügt; eine vorhandene Fußzeile auf dem Bildschirm verschwindet nach unten aus dem Bildschirm.
Beide Funktionen bewirken lediglich Änderungen auf dem Bildschirm, nicht im Speicherinhalt des Computers.

Beispiel:

```
DIREKTER DIALOG          PROGRAMMFORM

PRINT CHR$(25)           100 PRINT CHR$(25)
```

BILDSCHIRMFENSTER

Format: CHR$ (143) : Fensterecke rechts unten markieren
CHR$ (15) : Fensterecke links oben markieren

Zweck: Definition eines verkleinerten Darstellungsfensters auf dem Bildschirm eines CBM-Computers.

Bemerkung: Zur Definition eines Darstellungsfensters muß der Kursor in der Parameterliste einer PRINT-Anweisung zu den beiden gewünschten Fensterecken bewegt werden, wo mit den oben genannten Funktionen die Position der Fensterecke markiert wird.

```
DIREKTER DIALOG

?"■■■■■■";TAB(10);CHR$(15);"■■■■■■";TAB(60);CHR$(143)

PROGRAMMFORM

20 PRINT"■■■■■■";TAB(10);CHR$(15);"■■■■■■";TAB(60);CHR$(143)

BILDSCHIRMFENSTER ZWISCHEN ZEILEN 5 UND 10 UND SPALTEN
                                              15 UND 60
```

Zum Löschen eines Bildschirmfensters müssen zwei aufeinanderfolgende Tasteneingaben HOME in die Parameterliste einer PRINT-Anweisung aufgenommen werden.

```
DIREKTER DIALOG                    PROGRAMMFORM

?"■■"                              90 PRINT "■■"

"■"=SYMBOL FUER TASTE "HOME"
```

ZEICHENSATZ FÜR TEXT

Format: CHR$ (14)
oder: Taste ESC + Taste RVS + Taste "n"

Zweck: Umschalten auf den Zeichensatz für Text.

Bemerkung: Zur Umschaltung auf den Zeichensatz für Text wird die Funktion CHR$ (14) oder die gleichwertigen Tasteneingaben in die Parameterliste einer PRINT-Anweisung aufgenommen.

Beispiel:

```
DIREKTER DIALOG                    PROGRAMMFORM

PRINT CHR$(14)                     100 PRINT CHR$(14)
```

Anhang A

CBM Zeichen-Kodes

Dieser Anhang enthält folgende Tabellen:

- CBM ASCII-Kode zur Verschlüsselung von BASIC-Anweisungen: Tabelle A-1
- Standard-7-Bit-ASCII-Kode im externen Datenverkehr: Tabelle A-2
- 8-Bit CBM ASCII-Kode im Bildschirmspeicher: Tabelle A-3
- CBM ASCII-Kode im Arbeitsspeicher: Tabelle A-4

Die Tabellen A-1, A-2 und A-3 werden in Kapitel 7 bei der Beschreibung des Aufbaus von CBM-Computer-Systemen erklärt.

Tabelle A-4 enthält die Zusammenstellung der beiden, auf den Tastaturen von CBM-Computern verfügbaren Zeichensätze, ihre dezimale und hexadezimale Verschlüsselung im CBM ASCII-Kode sowie die ihnen beim Lesen oder Abspeichern mit den Befehlen PEEK oder POKE zugeordneten Dezimalziffern.

Standard-Zeichensatz. Der Standard-Zeichensatz wird durch Eingabe der Anweisung POKE 59468,12 aktiviert. Auf der Tastatur stehen dann die Großbuchstaben A - Z, bei Betätigung der Umschalttaste die graphischen Symbole, sowie die Zahlen 1 - 0 und Sonderzeichen zur Verfügung.

Alternativ-Zeichensatz. Der Alternativ-Zeichensatz wird mit der Anweisung POKE 59468,14 aktiviert. Auf der Tastatur stehen die Kleinbuchstaben a - z, und nach Bedienung der Umschalttaste die Großbuchstaben A - Z, sowie die Zahlen 1 - 0 und einige Sonderzeichen zur Verfügung.

CBM ASCII-Kode. Commodore Business Machines erweiterte den in den USA eingeführten ASCII-Kode (ASCII = American Standard Code for Information Interchange), um die für CBM-Computer kennzeichnenden Sonderzeichen wie z.B. graphische Symbole darzustellen.

Beim Arbeiten mit den Funktionen ASC() oder CHR#() verwenden Sie den Dezimalwert der in Spalte CBM ASCII angegebenen Verschlüsselung des gewünschten Zeichens. Im letzten Teil der Tabelle sind Zeichen in Negativschrift aufgeführt, die nicht durch CBM ASCII-Verschlüsselungen, sondern nur durch die in PEEK/POKE-Anweisungen verwendeten Dezimalziffern verschlüsselt sind.

PEEK/POKE. Beim Arbeiten mit den Anweisungen PEEK und POKE werden die in der Tabelle angegebenen Dezimalzahlen zur Darstellung der Zeichen verwendet. Ein Beispiel: Beim Lesen eines Speicherinhalts mit PEEK wird eine Dezimalzahl zurückgemeldet, aus der über die Spalte PEEK/POKE in Tabelle A-4 das gespeicherte Zeichen entschlüsselt werden kann.

Tabelle A-1. CBM ASCII-Kode zur Verschlüsselung von BASIC-Anweisungen*

Dezimal-wert	Zeichen/ Erklärung	Dezimal-wert	Zeichen/ Erklärung	Dezimal-wert	Zeichen/ Erklärung	Dezimal-wert	Zeichen/ Erklärung
0	Zeilenende	70	F	141	GOSUB	181	INT
1-31	———	71	G	142	RETURN	182	ABS
32	Leerstelle	72	H	143	REM	183	USR
33	!	73	I	144	STOP	184	FRE
34	"	74	J	145	ON	185	POS
35	#	75	K	146	WAIT	186	SQR
36	$	76	L	147	LOAD	187	RND
37	%	77	M	148	SAVE	188	LOG
38	&	78	N	149	VERIFY	189	EXP
39	'	79	O	150	DEF	190	COS
40	(	80	P	151	POKE	191	SIN
41	)	81	Q	152	PRINT#	192	TAN
42	*	82	R	153	PRINT	193	ATN
43	+	83	S	154	CONT	194	PEEK
44	,	84	T	155	LIST	195	LEN
45	—	85	U	156	CLR	196	STR$
46	.	86	V	157	CMD	197	VAL
47	/	87	W	158	SYS	198	ASC
48	0	88	X	159	OPEN	199	CHR$
49	1	89	Y	160	CLOSE	200	LEFT$
50	2	90	Z	161	GET	201	RIGHT$
51	3	91	[	162	NEW	202	MID$
52	4	92	\	163	TAB(	203	———
53	5	93	]	164	TO	204	CONCAT†
54	6	94	↑	165	FN	205	DOPEN†
55	7	95	←	166	SPC(	206	DCLOSE†
56	8	96-127	———	167	THEN	207	RECORD†
57	9	128	END	168	NOT	208	HEADER†
58	:	129	FOR	169	STEP	209	COLLECT†
59	;	130	NEXT	170	+	210	BACKUP†
60	<	131	DATA	171	-	211	COPY†
61	=	132	INPUT#	172	*	212	APPEND†
62	>	133	INPUT	173	/	213	DSAVE†
63	?	134	DIM	174	↑	215	CATALOG†
64	@	135	READ	175	AND	216	RENAME†
65	A	136	LET	176	OR	217	SCRATCH†
66	B	137	GOTO	177	>	218	DIRECTORY†
67	C	138	RUN	178	=	219	?SYNTAX ERROR†
68	D	139	IF	179	<	220-254	———
69	E	140	RESTORE	180	SGN	255	π

† Nur in CBM BASIC 4.0 * Siehe Kapitel 7

Tabelle A-2. Standard-7-Bit-ASCII-Kode im externen Datenverkehr

3	2	1	0	6 / 5 / 4	0 / 0 / 0	0 / 0 / 1	0 / 1 / 0	0 / 1 / 1	1 / 0 / 0	1 / 0 / 1	1 / 1 / 0	1 / 1 / 1
0	0	0	0		NUL	DLE	SP	0	@	P	'	p
0	0	0	1		SOH	DC1	!	1	A	Q	a	q
0	0	1	0		STX	DC2	"	2	B	R	b	r
0	0	1	1		ETX	DC3	#	3	C	S	c	s
0	1	0	0		EOT	DC4	$	4	D	T	d	t
0	1	0	1		ENQ	NAK	%	5	E	U	e	u
0	1	1	0		ACK	SYN	&	6	F	V	f	v
0	1	1	1		BEL	ETB	`	7	G	W	g	w
1	0	0	0		BS	CAN	(	8	H	X	h	x
1	0	0	1		HT	EM	)	9	I	Y	i	y
1	0	1	0		LF	SUB	*	:	J	Z	j	z
1	0	1	1		VT	ESC	+	;	K	[	k	{
1	1	0	0		FF	FS	, •	<	L	\	l	|
1	1	0	1		CR	GS	—	=	M	]	m	}
1	1	1	0		SO	RS	.	>	N	^	n	~
1	1	1	1		SI	US	/	?	O	—	o	DEL

NUL	Null	SO	Umschalttaste AUS	ESC	Umschaltzeichen
SOH	Anfang des Kopfes	SI	Umschalttaste EIN	FS	Hauptgruppen-Trennsymbol
STX	Textbeginn	DLE	Datenübertragungs-umschaltung	GS	Gruppen-Trennsymbol
ETX	Textende			RS	Untergruppen-Trennsymbol
EOT	Übertragungsende	DC1	Gerätekontrolle 1	US	Teilgruppen-Trennsymbol
ENQ	Anfrage	DC2	Gerätekontrolle 2	SP	Zwischenraum
ACK	Bestätigung	DC3	Gerätekontrolle 3	DEL	Löschen
BEL	Signal	DC4	Gerätekontrolle 4		
BS	Rücktaste	NAK	Negative Rückmeldung		
HT	Horizontales Tabulieren	STN	Synchronisierung		
LF	Zeilenvorschub	ETB	Ende des Übertragungsblocks		
VT	Vertikales Tabulieren	CAN	Ungültig		
FF	Formblatt-Vorlauf	EM	Ende der Aufzeichnung		
CR	Wagenrücklauf	SUB	Substitution		

Tabelle A-3. 8-Bit CBM ASCII-Kode im Bildschirmspeicher*

B 6 / 1 5 / 1 4 / 3210	0 / 0 / 0	0 / 0 / 1	0 / 1 / 0	0 / 1 / 1	1 / 0 / 0	1 / 0 / 1	1 / 1 / 0	1 / 1 / 1
0000	@	P		0	—	|		┌
0001	A	Q	!	1	♠	●	▌	┴
0010	B	R	"	2	|	—	▄	┬
0011	C	S	#	3	—	♥		┤
0100	D	T	$	4	¯	|	_	▌
0101	E	U	%	5	`	╭	|	▐
0110	F	V	&	6	—	╳	▒	▐
0111	G	W	'	7	|	○	|	▔
1000	H	X	(	8	|	♣	▒	▔
1001	I	Y	)	9	╮	|	▼	▄
1010	J	Z	*	:	╰	♦	|	┘
1011	K	L	+	;	╯	+	├	▪
1100	L	\	,	<	└	▓	▪	▘
1101	M	┘	-	=	\	|	└	┘
1110	N	↑	.	>	/	π	┐	▘
1111	O	←	/	?	|	◥	▁	▚

* Bit 8 ist nicht angegeben; es steuert die Umschaltung zwischen Normal- und Negativschrift (0 = Normalschrift, 1 = Negativschrift)

Tabelle A-4. CBM ASCII-Kode im Arbeitsspeicher

Standard-Zeichensatz	Alternativ-Zeichensatz	CBM ASCII DEC	CBM ASCII HEX	PEEK/POKE
S	A			
		0	00	
		1	01	
		2	02	
STOP	STOP	3	03	
		4	04	
		5	05	
		6	06	
		7	07	
		8	08	
		9	09	
		10	0A	
		11	0B	
		12	0C	
RETURN	RETURN	13	0D	
		14	0E	
		15	0F	
		16	10	
CRSR↓	CRSR↓	17	11	
RVS	RVS	18	12	
HOME	HOME	19	13	
DELETE	DELETE	20	14	
		21	15	
		22	16	
		23	17	
		24	18	
		25	19	
		26	1A	
		27	1B	
		28	1C	
CRSR→	CRSR→	29	1D	
		30	1E	
		31	1F	
ß	ß	32	20	32
!	!	33	21	33
"	"	34	22	34
#	#	35	23	35
$	$	36	24	36
%	%	37	25	37
&	&	38	26	38
'	'	39	27	39
(	(	40	28	40
)	)	41	29	41
*	*	42	2A	42
+	+	43	2B	43
,	,	44	2C	44
-	-	45	2D	45
.	.	46	2E	46
/	/	47	2F	47
0	0	48	30	48
1	1	49	31	49
2	2	50	32	50
3	3	51	33	51
4	4	52	34	52
5	5	53	35	53
6	6	54	36	54
7	7	55	37	55
8	8	56	38	56
9	9	57	39	57
:	:	58	3A	58
;	;	59	3B	59
<	<	60	3C	60
=	=	61	3D	61
>	>	62	3E	62
?	?	63	3F	63
@	@	64	40	0

Standard-Zeichensatz	Alternativ-Zeichensatz	CBM ASCII DEC	CBM ASCII HEX	PEEK/POKE
S	A			
A	a	65	41	1
B	b	66	42	2
C	c	67	43	3
D	d	68	44	4
E	e	69	45	5
F	f	70	46	6
G	g	71	47	7
H	h	72	48	8
I	i	73	49	9
J	j	74	4A	10
K	k	75	4B	11
L	l	76	4C	12
M	m	77	4D	13
N	n	78	4E	14
O	o	79	4F	15
P	p	80	50	16
Q	q	81	51	17
R	r	82	52	18
S	s	83	53	19
T	t	84	54	20
U	u	85	55	21
V	v	86	56	22
W	w	87	57	23
X	x	88	58	24
Y	y	89	59	25
Z	z	90	5A	26
[	[	91	5B	27
\	\	92	5C	28
]	]	93	5D	29
↑	↑	94	5E	30
←	←	95	5F	31
		96	60	32
!	!	97	61	33
"	"	98	62	34
#	#	99	63	35
$	$	100	64	36
%	%	101	65	37
&	&	102	66	38
'	'	103	67	39
(	(	104	68	40
)	)	105	69	41
*	*	106	6A	42
+	+	107	6B	43
,	,	108	6C	44
-	-	109	6D	45
.	.	110	6E	46
/	/	111	6F	47
0	0	112	70	48
1	1	113	71	49
2	2	114	72	50
3	3	115	73	51
4	4	116	74	52
5	5	117	75	53
6	6	118	76	54
7	7	119	77	55
8	8	120	78	56
9	9	121	79	57
:	:	122	7A	58
;	;	123	7B	59
<	<	124	7C	60
=	=	125	7D	61
>	>	126	7E	62
?	?	127	7F	63
		128	80	64

S = POKE 59468,12 A = POKE 59468,14 (s. Kapitel 1)

Tabelle A-4. CBM ASCII-Kode im Arbeitsspeicher (Forts.)

Standard-Zeichensatz	Alternativ-Zeichensatz	CBM ASCII DEC	CBM ASCII HEX	PEEK/POKE
S	A			
		129	81	65
		130	82	66
RUN	RUN	131	83	67
		132	84	68
		133	85	69
		134	86	70
		135	87	71
		136	88	72
		137	89	73
		138	8A	74
		139	8B	75
		140	8C	76
SHIFT+RETURN	SHIFT+RETURN	141	8D	77
		142	8E	78
		143	8F	79
		144	90	80
CRSR↑ RVS AUS CLR SCHIRM INSERT	CRSR↑ RVS AUS CLR SCHIRM INSERT	145	91	81
		146	92	82
		147	93	83
		148	94	84
		149	95	85
		150	96	86
		151	97	87
		152	98	88
		153	99	89
		154	9A	90
		155	9B	91
		156	9C	92
CRSR—	CRSR—	157	9D	93
		158	9E	94
		159	9F	95
SHIFT+Leertaste		160	A0	96
▮	▮	161	A1	97
▬	▬	162	A2	98
▔	▔	163	A3	99
_	_	164	A4	100
▏	▏	165	A5	101
▒	▒	166	A6	102
▕	▕	167	A7	103
▒	▒	168	A8	104
◤	▚	169	A9	105
▕	▕	170	AA	106
├	├	171	AB	107
▪	▪	172	AC	108
└	└	173	AD	109
┐	┐	174	AE	110
▁	▁	175	AF	111
┌	┌	176	B0	112
┴	┴	177	B1	113
┬	┬	178	B2	114
┤	┤	179	B3	115
▕	▕	180	B4	116
▌	▌	181	B5	117
▐	▐	182	B6	118
▔	▔	183	B7	119
▔	▔	184	B8	120
▬	▬	185	B9	121
▟	╱	186	BA	122
▪	▪	187	BB	123
▪	▪	188	BC	124
▗	▗	189	BD	125
▪	▪	190	BE	126
▚	▚	191	BF	127
─	─	192	C0	64

Standard-Zeichensatz	Alternativ-Zeichensatz	CBM ASCII DEC	CBM ASCII HEX	PEEK/POKE
S	A			
♠	A	193	C1	65
▕	B	194	C2	66
┬	C	195	C3	67
▔	D	196	C4	68
▔	E	197	C5	69
▁	F	198	C6	70
▕	G	199	C7	71
▕	H	200	C8	72
▏	I	201	C9	73
▏	J	202	CA	74
╱	K	203	CB	75
└	L	204	CC	76
╲	M	205	CD	77
╱	N	206	CE	78
┌	O	207	CF	79
┐	P	208	D0	80
●	Q	209	D1	81
▔	R	210	D2	82
♣	S	211	D3	83
▕	T	212	D4	84
▏	U	213	D5	85
╳	V	214	D6	86
○	W	215	D7	87
┼	X	216	D8	88
▕	Y	217	D9	89
♦	Z	218	DA	90
+	+	219	DB	91
▒	▒	220	DC	92
▕	▕	221	DD	93
π	▒	222	DE	94
◥	▒	223	DF	95
b	b	224	E0	96
▮	▮	225	E1	97
▬	▬	226	E2	98
▔	▔	227	E3	99
_	_	228	E4	100
▏	▏	229	E5	101
▒	▒	230	E6	102
▕	▕	231	E7	103
▒	▒	232	E8	104
◤	▚	233	E9	105
▕	▕	234		106
├	├	235		107
▪	▪	236		108
└	└	237		109
┐	┐	238		110
▁	▁	239		111
┌	┌	240		112
┴	┴	241		113
┬	┬	242		114
┤	┤	243		115
▕	▕	244		116
▌	▌	245		117
▐	▐	246		118
▔	▔	247		119
▔	▔	248		120
▬	▬	249		121
▟	╱	250		122
▪	▪	251		123
▪	▪	252		124
▗	▗	253		125
▪	▪	254		126
π	▒	255		127

S = POKE 59468,12 A = POKE 59468,14 (s. Kapitel 1)

Tabelle A-4. CBM ASCII-Kode im Arbeitsspeicher (Forts.)

Standard-Zeichensatz S	Alternativ-Zeichensatz A	CBM ASCII DEC	CBM ASCII HEX	PEEK/POKE
				128
	a			129
	b			130
	c			131
	d			132
	e			133
	f			134
	g			135
	h			136
	i			137
	j			138
	k			139
	l			140
	m			141
	n			142
	o			143
	p			144
	q			145
	r			146
	s			147
	t			148
	u			149
	v			150
	w			151
	x			152
	y			153
	z			154
				155
				156
				157
				158
				159
				160
				161
				162
				163
				164
				165
				166
				167
				168
				169
				170
				171
				172
				173
				174
				175
				176
				177
				178
				179
				180
				181
				182
				183
				184
				185
				186
				187
				188
				189
				190
				191

in Negativschrift

Standard-Zeichensatz S	Alternativ-Zeichensatz A	CBM ASCII DEC	CBM ASCII HEX	PEEK/POKE
				192
				193
				194
				195
				196
				197
				198
				199
				200
				201
				202
				203
				204
				205
				206
				207
				208
				209
				210
				211
				212
				213
				214
				215
				216
				217
				218
				219
				220
				221
				222
				223
				224
				225
				226
				227
				228
				229
				230
				231
				232
				233
				234
				235
				236
				237
				238
				239
				240
				241
				242
				243
				244
				245
				246
				247
				248
				249
				250
				251
				252
				253
				254
				255

S = POKE 59468,12 A = POKE 59468,14 (s. Kapitel 1)

Anhang B

CBM Fehler-Meldungen

Fehlermeldungen können im Zusammenhang mit fast jeder Tasteneingabe und jedem Programmlauf auf dem Bildschirm ausgegeben werden. Sowohl der CBM BASIC-Übersetzer als auch das Betriebssystem können Fehlermeldungen ausgeben, die im folgenden aufgeführt sind.

Entdeckt der CBM BASIC-Übersetzer einen Fehler, dann stellt er seine Diagnose in Form einer Meldung dar, die mit einem Fragezeichen beginnt:

? fehlerart ERROR IN LINE zeilennummer

Neben dem unveränderlichen Teil der Meldung ERROR IN LINE (Fehler auf Programmzeile) erscheinen links die "fehlerart" und rechts die "zeilennummer" derjenigen Programmzeile, in der ein Fehler entdeckt wurde. Die möglichen Meldungen der "fehlerart" sind unten alphabetisch aufgeführt. Die Angabe einer "zeilennummer" entfällt beim Arbeiten in direktem Dialog. Nach jeder Fehlermeldung kehrt der Computer zum direkten Dialog zurück und zeigt mit der Nachricht READY an, daß er für Benutzereingaben bereit ist.

CBM BASIC-FEHLERMELDUNGEN

Fehlerart	Ursache und mögliche Abhilfe
BAD SUBSCRIPT	Beim Arbeiten mit einer Tabelle wurde der Versuch gemacht, einen Tabelleneintrag anzusprechen, der außerhalb des Wertebereichs der Indices lag, mit denen die Tabelle definiert wurde (DIM-Anweisung), oder eine Anzahl von Dimensionen enthielt, die nicht mit den Angaben in der zugehörigen DIM-Anweisung übereinstimmt, oder, bei fehlender DIM-Anweisung, einen Index-Wert größer 10 verwendete. Ändern Sie zur Abhilfe den Indexwert des Tabelleneintrags oder die Angaben in der DIM-Anweisung.
CAN'T CONTINUE	Ein Programmablauf kann nach Eingabe der Anweisung CONT nicht fortgesetzt werden, da das Programm in direktem Dialog verändert wurde, oder der Pro-

Fehlerart	Ursache und mögliche Abhilfe
	grammablauf im Zusammenhang mit einer Fehlermeldung unterbrochen wurde. Korrigieren Sie zur Abhilfe den Programmfehler und geben Sie, wenn möglich, RUN ein, um das Programm von vorne zu starten oder mit einem GOTO an der Unterbrechungsstelle fortzusetzen.
DIVISION BY ZERO	Diese Fehlermeldung ist die Antwort auf eine nicht zulässige Division durch Null. Prüfen Sie zur Abhilfe die Zahlenwerte der Variablen oder Konstanten in der angegebenen Zeilennummer. Ändern Sie das Programm derart, daß ein Divisor niemals den Wert 0 annehmen kann oder fügen Sie einen Null-Test vor jeder Division ein.
FORMULA TOO COMPLEX	Diese Meldung zeigt keinen Programmfehler an, sondern erscheint, wenn die in einer Programmzeile geforderten Operationen zu komplex für CBM BASIC sind. Teilen Sie die beanstandete Programmzeile in zwei oder mehrere Ausdrücke und wiederholen Sie den Programmlauf.
ILLEGAL DIRECT	In direktem Dialog wurde eine Anweisung gegeben, die nur in Programmform zulässig ist, wie: DATA, DEF FN, GET, GET#, INPUT, INPUT#. Geben Sie zur Abhilfe die gewünschte Operation in Form eines kurzen Programms ein.
ILLEGAL QUANTITY	Einer Funktion wurden einer oder mehrere Parameter übergeben, die außerhalb des zulässigen Wertebereichs lagen. Diese Meldung erfolgt auch, wenn mit USR eine Funktion angesprochen wird, noch ehe die Unterprogramm-Adresse in den Speicherplätzen 1 und 2 abgelegt wurden. Prüfen Sie zur Abhilfe die Wertebereiche von Parametern mit Hilfe der Angaben in Kapitel 8. Ändern Sie das Programm derart, daß ein Argument immer im zulässigen Wertebereich liegt, oder fügen Sie eine Prüfung von Argumenten vor jeder Funktion ein. Bei einem USR-Fehler fügen Sie POKE-Anweisungen ein, mit denen die Unterprogramm-Adresse vor Ansprechen von USR gespeichert wird.
NEXT WITHOUT FOR	Im Programm wurde eine NEXT-Anweisung ohne vorangegangene FOR-Anweisung gefunden. Entweder war keine FOR-NEXT-Anweisung vorgesehen, oder die Variable nach NEXT stimmt nicht mit der Variablen nach FOR überein.
OUT OF DATA	Bei Ausführung einer READ-Anweisung wird festgestellt, daß alle Angaben in den DATA-Anweisungen bereits gelesen wurden. Jeder Variablen in READ muß ein Element in DATA entsprechen. Erweitern Sie zur Abhilfe die Zahl der Elemente in DATA oder beschränken Sie die Zahl der Variablen in READ auf

Fehlerart	Ursache und mögliche Abhilfe
	die augenblickliche Zahl von Elementen in DATA. Fügen Sie eine RESTORE-Anweisung ein, um die Elemente von DATA erneut zu lesen; oder fügen Sie als letztes Element in DATA eine "Flagge" ein, nach deren Lesen keine weiteren READ-Anweisungen mehr ausgeführt werden. (Jedes nicht als DATA-Element verwendetes Zeichen kann als Flagge dienen.
OUT OF MEMORY	Der Speicherbereich für Benutzerprogramme (siehe Kapitel 7) war bereits belegt, als die Speicherung z.B. einer weiteren Programmzeile versucht wurde. Diese Meldung kann auch durch mehrfache FOR-NEXT- und/oder verschachtelte GOSUB-Anweisungen ausgelöst worden sein, die den Stapelspeicher belegen; dies ist der Fall, wenn ?FRE(0) noch freien Speicherbereich für Programme anzeigt. Vereinfachen Sie zur Abhilfe das Programm; verringern insbesondere die Größe von Tabellen (Feldern). Möglicherweise müssen Sie das Programm segmentieren und die Segmente von einem externen Speicher nacheinander laden. (Siehe Kapitel 6).
OVERFLOW	Das Ergebnis einer Berechnung liegt außerhalb des zulässigen Wertebereichs, d.h. die Zahl ist zu groß. Die größte im Computer zulässige Zahl ist 1.70141184E+38 Zur Abhilfe kann es möglich sein, Wertüberschreitungen durch Änderung der Reihenfolge von Berechnungen zu vermeiden.
REDIM'D ARRAY	Der Name für eine Tabelle (Feld) erscheint in mehr als einer DIM-Anweisung. Diese Meldung erscheint auch, wenn der Tabellenname zunächst ohne DIM verwendet wurde, d.h. mit elf Eintragungen, und später in einer DIM-Anweisung auftritt. Setzen Sie zur Abhilfe DIM-Anweisungen an den Programmanfang, um leicht prüfen zu können, daß jede DIM-Anweisung nur einmal ausgeführt wird. DIM darf nicht innerhalb einer FOR-NEXT-Schleife oder in einem Unterprogramm verwendet werden, wo Gefahr besteht, daß DIM mehr als einmal ausgeführt wird.
REDO FROM START	Diese Meldung bedeutet keinen schwerwiegenden Fehler und tritt während der Ausführung einer INPUT-Anweisung auf, wenn die Eingabeaufforderung von INPUT mit einem falschen Datentyp beantwortet wurde (z.B. Zeichenkette statt erwarteter numerischer Variablen). Geben Sie als Abhilfe Daten vom richtigen Typ ein.
RETURN WITHOUT GOSUB	Im Programm wurde eine RETURN-Anweisung ohne vorangehende, zugehörige GOSUB-Anweisung angetroffen. Löschen Sie als Abhilfe die RETURN-Anweisung oder tragen Sie die fehlende GOSUB-Anweisung

Fehlerart	Ursache und mögliche Abhilfe
	nach. Der Fehler kann durch unbeabsichtigte Ausführung eines Unterprogramms entstanden sein. Korrigieren Sie in diesem Fall den Flußplan des Programms und setzen Sie zur Unterstützung der Fehlersuche eine END- oder STOP-Anweisung unmittelbar vor den Anfang eines Unterprogramms.
STRING TOO LONG	Mit Hilfe des Verkettungs-Operators "+" wurde versucht, eine Zeichenkette mit mehr als den zulässigen 255 Zeichen zu bilden. Lösen Sie zur Abhilfe die Zeichenkette in zwei oder mehr kürzere Zeichenketten auf und prüfen Sie vor Verkettungen mit der Funktion LEN die Länge von Zeichenketten.
SYNTAX	In der gerade im direkten Dialog eingegebenen oder bei Programmausführung angesprochenen Programmzeile wurde ein Syntax-Fehler festgestellt. Diese häufigste unter den Fehlermeldungen, entsteht durch Schreibfehler bei BASIC-Anweisungen, falscher Zeichensetzung, nicht paarweisen Klammern, unzulässigen Zeichen usw. Beachten Sie, daß Syntax-Fehler erst beim Programmlauf, nicht schon bei der Eingabe festgestellt werden.
TYPE MISMATCH	Es wurde versucht, einer numerischen Variablen eine Zeichenkette zuzuordnen, oder umgekehrt, oder das Argument einer Funktion wurde im falschen Datentyp angegeben. Ändern Sie zur Abhilfe den Datentyp der beanstandeten Größe. Siehe Kapitel 8 für zulässige Datentypen.
UNDEF'D STATEMENT	Im Programm wurde der Versuch gemacht, zu einer nicht bestehenden Zeilennummer zu verzweigen. Fügen Sie zur Abhilfe eine Anweisung mit der genannten Zeilennummer ein oder verzweigen Sie zu einer anderen Zeilennummer.
UNDEF'D FUNCTION	Im Programm wurde eine vom Benutzer definierte Funktion angesprochen, die nicht zuvor mit einer DEF FN-Anweisung erklärt wurde. Erklären Sie die Funktion mit DEF FN am Programmanfang.

FEHLERMELDUNGEN DES BETRIEBSSYSTEMS

Fehlerart	Ursache und mögliche Abhilfe
BAD DATA	Statt der erwarteten numerischen Daten wurde eine Zeichenkette eingegeben. Ändern Sie zur Abhilfe die Eingabe auf numerische Daten oder ändern Sie das

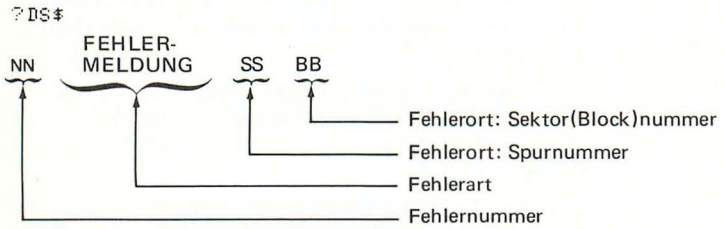
Fehlerart	Ursache und mögliche Abhilfe
	Programm derart, daß es die Eingabe von Zeichenketten zuläßt.
BAD DISK	Bei Ausführung einer HEADER-Anweisung wurde ein Fehlverhalten der Diskette registriert, ausgelöst durch Floppy-Disk-Speicher ohne eingelegte Diskette, Disketten mit Schreibschutz, fehlerhaft formatierte Disketten oder eine defekte magnetische Oberfläche. Überprüfen Sie zur Abhilfe den Disketten-Speicher und die Diskette oder tauschen Sie diese aus.
DEVICE NOT PRESENT	Das Ansprechen eines externen Gerätes über den IEEE 488-Bus blieb ohne Rückantwort. Das Statusregister hat nach Ablauf der Wartezeit den Wert 2 angenommen. Diese Fehlermeldung kann bei jeder Ein/Ausgabe-Anweisung auftreten. Überprüfen Sie zur Abhilfe, ob die OPEN-Anweisung die richtige Gerätenummer angibt. War die beanstandete Anweisung zuvor ausführbar, prüfen Sie das angesprochene Gerät auf Fehlfunktion, Fehlanschluß oder Betriebszustand Ein/Aus.
FILE EXISTS	Beim Kopieren einer Quelldatei mit der Anweisung COPY wird festgestellt, daß diese Datei bereits auf der Ziel-Diskette existiert. Löschen Sie zur Abhilfe die Datei auf der Ziel-Diskette oder verwenden Sie eine andere Diskette als Ziel-Diskette.
FILE NOT FOUND	In einer LOAD- oder OPEN-Anweisung wurde ein Dateiname angegeben, der auf dem angegebenen Gerät nicht gefunden werden konnte. Prüfen Sie zur Abhilfe, ob Sie das Speichergerät mit der richtigen Kassette oder Diskette geladen haben. Prüfen Sie die Dateinamen auf der Kassette oder Diskette nach möglichen Schreibfehlern.
FILE NOT OPEN	Es wurde der Zugriff auf eine Datei versucht, die nicht durch eine OPEN-Anweisung zuvor geöffnet wurde. Die Abhilfe besteht im Öffnen der Datei mit OPEN.
FILE OPEN	Es wurde das Öffnen einer Datei versucht, die bereits durch eine vorausgehende OPEN-Anweisung geöffnet worden war. Prüfen Sie zur Abhilfe die logische Dateinummer in den OPEN-Anweisungen, um sicherzustellen, daß für jede Datei eine verschiedene Dateinummer verwendet wurde. Fügen Sie eine CLOSE-Anweisung ein, ehe Sie die gleiche Datei für eine Ein/Ausgabe-Operation erneut öffnen wollen.
LOAD	Beim Laden eines Programms von der Kassette ist eine unzulässige Zahl von Bandlesefehlern aufgetreten (mehr als 31), die nicht gelöscht wurden. Diese Fehler-

Fehlerart	Ursache und mögliche Abhilfe
	meldung wird im Zusammenhang mit der Anweisung LOAD ausgegeben.
NOT INPUT FILE	Es wurde das Lesen einer Datei auf Kassette versucht, die nur zum Schreiben geöffnet worden ist. Prüfen Sie als Abhilfe die Parameter in den Anweisungen READ# und OPEN. Zum Lesen von einer Kassetten-Datei muß der dritte Parameter in OPEN 0 sein.
NOT OUTPUT FILE	Es wurde versucht, auf eine Kassetten-Datei zu schreiben, die nur zum Lesen geöffnet worden war. Prüfen Sie zur Abhilfe die Parameter in den Anweisungen PRINT# und OPEN. Beim Schreiben auf eine Datei ist als dritter Parameter in OPEN eine 1 (oder eine 2, wenn eine Bandendemarke EOT gewünscht ist) erforderlich.
VERIFY ERROR	Bei Ausführung der Anweisung VERIFY wird festgestellt, daß das Programm im Speicher des Computers und das zu prüfende Programm auf einem externen Speicher nicht übereinstimmen.

FEHLERMELDUNGEN DES DISKETTEN-BETRIEBS-SYSTEMS (DOS)

ANFORDERUNG VON FEHLERMELDUNGEN

Führen Sie in BASIC 4.0 zur Anforderung und Bildschirmdarstellung von Fehlermeldungen eine PRINT-Anweisung mit den Variablen DS oder DS$ aus. Die Fehlermeldung ist in DS$ wie folgt verschlüsselt:



Im BASIC < 3.0 ist kein Zugriff auf die Variablen DS oder DS$ möglich. Zur Untersuchung des Fehlerstatus eines Floppy Disk-Speichers müssen Sie eine logische Datei mit der Anweisung OPEN öffnen, in der als Geräteadresse 8 und als Sekundäradresse 15 angegeben sind. Danach müssen Sie vier Zeichenketten-Variable lesen und auf dem Bildschirm darstellen:

```
10 OPEN 1,8,15
20 INPUT#1, A$, B$, C$, D$
30 PRINT A$,B$,C$,D$
40 CLOSE 1
```

A$ ist die Nummer der Fehlermeldung, B$ enthält die Fehlermeldung, C$ gibt die Nummer der Spur an und D$ die Nummer des Sektors.

Tabelle B-1 enthält eine Zusammenstellung der Spurnummern und Sektornummern aller DOS-Fehlermeldungen.

Tabelle B-1. Fehlermeldungen des Betriebssystems für Disketten, DOS

	Fehler # NN	Fehlermeldung	Fehlerart	Spur # SS	Sektor # BB
Status-Meldungen	00	OK	Dateien gelöscht	00	00
	01	FILES SCRATCHED		SS*	00
Lese-Fehler	20	READ ERROR	Block-Kopfeintrag nicht gefunden	SS+	BB+
	21	READ ERROR	Kein Synchronisierzeichen	SS	BB
	22	READ ERROR	Datenblock nicht vorhanden	SS	BB
	23	READ ERROR	Prüfsummenfehler im Datenblock	SS	BB
	24	READ ERROR	Fehlerhafte Byte-Dekodierung	SS	BB
	27	READ ERROR	Prüfsummenfehler im Kopfeintrag	SS	BB
Schreib-Fehler	25	WRITE ERROR	Schreib-/Prüf-Fehler	SS	BB
	26	WRITE PROTECT ON	Schreibschutz wirksam	SS	BB
	28	WRITE ERROR	Datenblock zu lang	SS	BB
	29	DISK ID MISMATCH	Abweichende Disketten-Kennzahl	SS	BB
Syntax-Fehler	30	SYNTAX ERROR	Allgemeine Syntax	00	00
	31	SYNTAX ERROR	Ungültige Anweisung	00	00
	32	SYNTAX ERROR	Zu lange Anweisung	00	00
	33	SYNTAX ERROR	Ungültiger Dateiname	00	00
	34	SYNTAX ERROR	Dateiname fehlt	00	00
	39	SYNTAX ERROR	Ungültige DOS-Anweisung	00	00
	50	SYNTAX ERROR	Aufzeichnung nicht vorhanden	SS	BB
	51	SYNTAX ERROR	Aufzeichnung zu kurz; Überlauf	SS	BB
	52	SYNTAX ERROR	Disketten-Überlauf	SS	BB
Datei-Fehler	60	WRITE FILE OPEN	Datei bereits geöffnet z. Schreiben	00	00
	61	FILE NOT OPEN	Datei nicht geöffnet	00	00
	62	FILE NOT FOUND	Datei nicht gefunden	00	00
	63	FILE EXISTS	Datei existierte bereits	00	00
	64	FILE TYPE MISMATCH	Dateityp-Angaben widersprüchlich	00	00
	65	NO BLOCK	Kein freier Block (Sektor)	SS	BB
	66	ILLEGAL TRACK AND SECTION	Spur und Sektor unzulässig	SS	BB
	67	ILLEGAL SYSTEM TRACK AND SECTOR	Spur und Sektor unzulässig	SS	BB
System-Fehler	70	NO CHANNEL	Kein freier Kanal	00	00
	71	DIR ERROR	Inhaltsverzeichnis unzutreffend	00	00
	72	DISK FULL	Diskette voll	00	00
	73	DOS MISMATCH	Abweichende DOS-Version	00	00
	74	DRIVE NOT READY	Laufwerk ohne Diskette	00	00

*) Die Spurnummer SS gibt die Anzahl gelöschter Dateien an.
+) Der Fehlerort wird in SS, BB mitgeteilt.

LESEFEHLER

Fehlernummer	Fehlermeldung	Fehlerursache
20	Kopfeintrag des Blocks nicht gefunden	Die Kontrolleinheit des Floppy Disk-Speichers kann den Kopfeintrag des gewünschten Datenblocks nicht finden. Ursache kann eine unzulässige Sektornummer oder die Löschung des Kopfeintrags sein.

Fehlernummer	Fehlermeldung	Fehlerursache
21	Synchronisier-Marke nicht gefunden	Die Kontrolleinheit des Floppy Disk-Speichers kann auf der gewünschten Spur keine Synchronisier-Marke finden. Ursache kann eine Dejustierung des Schreib/Lese-Kopfes, eine fehlende Diskette oder ein Gerätefehler sein.
22	Datenblock nicht vorhanden	Die Kontrolleinheit des Floppy Disk-Speichers wurde aufgefordert, einen Datenblock zu lesen oder zu prüfen, der nicht ordnungsgemäß aufgezeichnet wurde. Diese Fehlermeldung tritt im Zusammenhang mit BLOCK-Anweisungen auf und gibt einen unzulässigen Zugriff auf eine Spur und/oder einen Sektor an.
23	Prüfsummen-Fehler im Datenblock	Diese Fehlermeldung gibt an, daß ein oder mehrere Bytes der gelesenen Daten fehlerhaft ist. Die Daten wurden zwar in den Speicher des Betriebssystems DOS im Floppy Disk-Speicher gelesen, die Prüfsumme zeigt aber einen Fehler in den übertragenen Daten an. Diese Meldung kann auch Hinweis auf Erdungsprobleme sein.
24	Byte-Dekodier-Fehler	Daten oder ein Kopfeintrag wurden zwar in den DOS-Speicher gelesen, aufgrund eines Gerätefehlers ist jedoch ein fehlerhaftes Bit-Muster aufgetreten.
27	Prüfsummenfehler im Kopfeintrag	Die Kontrolleinheit im Floppy Disk-Speicher hat einen Fehler im Kopfeintrag des gewünschten Datenblocks festgestellt. Der Datenblock wurde nicht in den Speicher des Betriebssystems DOS gelesen. Auch diese Meldung kann ein Hinweis auf Erdungsprobleme sein.

SCHREIBFEHLER

Fehlernummer	Fehlermeldung	Fehlerursache
25	Schreib-Prüffehler	Diese Meldung wird ausgegeben, wenn die Kontrolleinheit im Floppy Disk-Speicher keine Übereinstimmung zwischen den aufgezeichneten und den im Speicher des Betriebssystems DOS gespeicherten Daten feststellt.
26	WRITE PROTECT ON	Diese Meldung wird ausgegeben, wenn die Kontrolleinheit des Floppy Disk-Speichers einen Datenblock schreiben sollte, obwohl

Fehlernummer	Fehlermeldung	Fehlerursache
		der Schalter für Schreibschutz betätigt war. Dieser Fall tritt ein, wenn an einer Diskette der Schreibschutz angebracht wurde.
28	Langer Datenblock	Die Kontrolleinheit des Floppy Disk-Speichers versucht, die Synchronisiermarke des nächsten Kopfeintrags zu entdecken, nachdem ein Datenblock geschrieben wurde. Wenn die Synchronisiermarke nicht innerhalb einer festgesetzten Zeit erscheint, wird eine Fehlermeldung erzeugt. Ursache des Fehlers ist ein schlechtes Format der Diskette (Daten erstrecken sich in den nächsten Block) oder ein Gerätefehler.
29	DISK ID	Die Kontrolleinheit des Floppy Disk-Speichers sollte auf eine nicht-initiierte Diskette zugreifen. Diese Fehlermeldung kann auch bei einer Diskette mit schlechtem Kopfeintrag ausgelöst werden.

SYNTAX-FEHLER

Fehlernummer	Fehlermeldung	Fehlerursache
30	Allgemeine Syntax	Das Betriebssystem für Disketten, DOS, kann die über den Befehlskanal gesandte Anweisung nicht interpretieren. Dies wird typisch verursacht durch eine unzulässige Zahl von Dateinamen oder unzulässige Anordnungen, wenn z.B. zwei Dateinamen auf der linken Seite einer COPY-Anweisung erscheinen.
31	Ungültige Anweisung	Das Betriebssystem DOS erkennt die Anweisung nicht an. Die Anweisung muß mit der ersten Stelle beginnen.
32	Zu lange Zeile	Die gesamte Anweisung ist länger als 40 Zeichen.
33	Ungültiger Dateiname	In einer OPEN- oder SAVE-Anweisung wurde ein bereits existierender Dateiname angegeben.
34	Keine Datei vorhanden	Der Dateiname wurde in der Anweisung vergessen oder das Betriebssystem DOS interpretiert ihn nicht als solchen. Typische Ursachen sind fehlende Anführungszeichen (") oder Doppelpunkte (:) in der Anweisung.

Fehlernummer	Fehlermeldung	Fehlerursache
39	Unzulässige DOS-Anweisung	Eine nicht interpretierte Anweisung an das Betriebssystem DOS wurde empfangen.
50	Aufzeichnung nicht vorhanden	Mit einer INPUT#- oder GET#-Anweisung wurde eine Aufzeichnung jenseits des Endes der augenblicklichen Datei angesprochen. Die Fehlermeldung entsteht, wenn die angesprochene Aufzeichnung gelesen werden soll; sie ist nicht notwendigerweise ein Fehler, wenn Sie an das Ende einer Datei positionieren, um neue Aufzeichnungen an die alte Datei anzufügen.
51	Überlauf der Aufzeichnung	Eine PRINT#-Anweisung versuchte mehr als die zulässige Zahl von Zeichen in eine Relative Datei zu schreiben. Der abschließende Wagenrücklauf (CR) wird als ein Zeichen in der Aufzeichnungslänge gezählt.
52	Datei zu groß	Die augenblickliche Aufzeichnungsstelle läßt erkennen, daß bei der nächsten Schreiboperation auf die Diskette ein Überlauf auftritt.

DATEIFEHLER

Fehlernummer	Fehlermeldung	Fehlerursache
60	WRITE FILE OPEN	Diese Fehlermeldung wird erzeugt, wenn eine zum Schreiben geöffnete Datei nicht geschlossen wurde, ehe sie zum Lesen erneut geöffnet wurde.
61	FILE NOT OPEN	Diese Fehlermeldung wird erzeugt, wenn auf eine Datei zugegriffen wurde, die nicht im Betriebssystem DOS geöffnet wurde. Manchmal wird in solchen Fällen keine Fehlermeldung erzeugt; die Aufforderung zum Zugriff wird in diesem Fall ignoriert.
62	FILE NOT FOUND	Die gewünschte Datei besteht nicht im angegebenen Laufwerk des Floppy Disk-Speichers.
63	FILE EXISTS	Der Dateiname einer neu angelegten Datei besteht bereits auf der Diskette.

Fehlernummer	Fehlermeldung	Fehlerursache
64	FILE TYPE MISMATCH	Der Dateityp der gewünschten Datei stimmt nicht mit dem Dateityp im Inhaltsverzeichnis überein.
65	NO BLOCK	Diese Fehlermeldung wird im Zusammenhang mit einer BLOCK ALLOCATE-Anweisung erzeugt und gibt an, daß der bezeichnete Block bereits zugewiesen wurde. Die Parameter in der Fehlermeldung geben Spur und Sektor an, die mit nächsthöherer Kennzahl verfügbar sind. Sind die Parameter Null, dann sind alle Blöcke mit höherer Kennzahl in Benutzung.
66	ILLEGAL TRACK OR SECTOR	Es wurde der Versuch gemacht, auf einen Sektor zuzugreifen, der physikalisch nicht existiert. Die Spur- und/oder Sektornummer in den Angaben liegt außerhalb des zulässigen Wertebereichs für die angesprochene Diskette. Sie sollten eine derartige Fehlermeldung nur bei Dateien mit wahlfreiem Zugriff sehen.
67	ILLEGAL SYSTEM TRACK AND SECTOR	Beim Zugriff auf Dateien für Programme oder Daten wurde der Versuch gemacht, auf einen Sektor zuzugreifen, der für das Disketten-Betriebssystem reserviert ist.

SYSTEMFEHLER

Fehlernummer	Fehlermeldung	Fehlerursache
70	NO CHANNEL	Der gewünschte Kanal ist nicht verfügbar oder alle Kanäle sind in Benutzung. Zur gleichen Zeit können max. 5 Sequentielle Dateien geöffnet werden. Kanäle für direkten Zugriff können bis zu sechs geöffnete Dateien haben.
71	DIR ERROR	Die Speicherbelegungsliste BAM stimmt nicht mit der internen Zählung überein. Das Problem liegt innerhalb der BAM-Zuordnung oder die Blockbelegungsliste BAM wurde im Speicher des Betriebssystems DOS überschrieben. Initiieren Sie die Diskette erneut, um die Blockbelegungsliste BAM im Speicher wieder anzulegen. Benutzte Dateien können durch diese Korrektur geschlossen werden.

Fehlernummer	Fehlermeldung	Fehlerursache
72	DISK FULL	Entweder sind alle Blöcke auf der Diskette in Benutzung, oder das Inhaltsverzeichnis läuft über (mehr als 152 Einträge).
73	DOS MISMATCH	Daten, die auf eine Diskette in irgendeiner Version des Betriebssystems DOS geschrieben wurden, können mit jeder anderen DOS-Version gelesen werden. Zum Schreiben auf die Diskette müssen Sie jedoch die gleiche DOS-Version verwenden, mit der die Diskette initiiert wurde. Fehlernummer 73 wird gemeldet, wenn Sie versuchen, beim Schreiben auf eine Diskette eine abweichende DOS-Version zu verwenden, als beim Anlegen und Initiieren der Diskette verwendet wurde.
74	DRIVE NOT READY	Es wurde versucht, auf eine Diskette im angegebenen Laufwerk eines CBM 8050 Floppy Disk-Speichers zuzugreifen.

Anhang C

Umwandlungs-Tabellen

Dieser Anhang enthält folgende Nachschlag-Tafeln:

- Hexadezimal-Dezimal-Umwandlung von Ganzzahlen
- Potenzen der Zahl 2
- Mathematische Konstanten
- Potenzen der Zahl 16
- Potenzen der Zahl 10 in Hexadezimal-Darstellung

HEXADEZIMAL-DEZIMAL-UMWANDLUNGEN

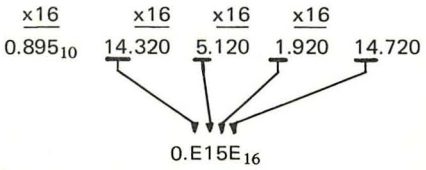
Die Tabelle auf den folgenden Seiten erlaubt eine unmittelbare Umwandlung zwischen Hexadezimalzahlen im Bereich 0 - FFF und Dezimalzahlen im Bereich 0 - 4095. Für die Umwandlung größerer Ganzzahlen addieren Sie zu den Tabellenwerten aus dem Bereich 0 - FFF bzw. 0 - 4095 Werte aus folgender Tabelle:

Hexadezimal	Dezimal	Hexadezimal	Dezimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Hexadezimale Bruchzahlen werden in folgenden Schritten in dezimale Bruchzahlen umgewandelt:

1. Drücken Sie die hexadezimale Bruchzahl als Ganzzahl, multipliziert mit 16^{-n}, aus. Hierbei stellt n die Anzahl der Stellen nach dem Hexadezimal-Punkt dar:
   $0.CA9BF3_{16} \Rightarrow CA9BF3_{16} \times 16^{-6}$
2. Finden Sie die Dezimaldarstellung dieser hexadezimalen Ganzzahl:
   $CA9BF3_{16} = 13\,278\,195_{10}$
3. Multiplizieren Sie die Dezimaldarstellung mit 16^{-n} (wofür dieses Kapitel eine Potenztabelle enthält):
   $13\,278\,195 \times 0.59604\,64477 \times 10^{-7}$
   $= 0.791\,442\,096_{10}$
   $= 0.CA9BF3_{16}$

Dezimale Bruchzahlen werden in hexadezimale Bruchzahlen durch wiederholte Multiplikationen des dezimalen Bruchanteils mit 16 und Umwandlung des entstehenden Ganzzahlanteils in Hexadezimalzahlen umgewandelt, wie folgendes Schema am Dezimalbruch 0.895 zeigt:



HEXADEZIMAL-DEZIMAL-UMWANDLUNG VON GANZZAHLEN

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

HEXADEZIMAL-DEZIMAL-UMWANDLUNG VON GANZZAHLEN

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

HEXADEZIMAL-DEZIMAL-UMWANDLUNG VON GANZZAHLEN

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1562	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

HEXADEZIMAL-DEZIMAL-UMWANDLUNG VON GANZZAHLEN

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

HEXADEZIMAL-DEZIMAL-UMWANDLUNG VON GANZZAHLEN

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

HEXADEZIMAL-DEZIMAL-UMWANDLUNG VON GANZZAHLEN

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

POTENZEN DER ZAHL 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125

MATHEMATISCHE KONSTANTEN

KONSTANTE	DEZIMALWERT	HEXADEZIMALWERT
π	3.14159 26535 89793	3.243F 6A89
π^{-1}	0.31830 98861 83790	0.517C C1B7
$\sqrt{\pi}$	1.77245 38509 05516	1.C5BF 891C
$\ln \pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.B7E1 5163
e^{-1}	0.36787 94411 71442	0.5E2D 58D9
$\sqrt{e}$	1.64872 12707 00128	1.A612 98E2
$\log_{10} e$	0.43429 44819 03252	0.6F2D EC55
$\log_2 e$	1.44269 50408 88963	1.7154 7653
γ	0.57721 56649 01533	0.93C4 67E4
$\ln \gamma$	-0.54953 93129 81645	-0.8CAE 9BC1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
$\ln 2$	0.69314 71805 59945	0.B172 17F8
$\log_{10} 2$	0.30102 99956 63981	0.4D10 4D42
$\sqrt{10}$	3.16227 76601 68379	3.2988 075C
$\ln 10$	2.30258 40929 94046	2.4D75 3777

POTENZEN DER ZAHL 16

16^n	n	16^{-n}
1	0	$0.10000\ 00000\ 00000\ 00000 \times 10$
16	1	$0.62500\ 00000\ 00000\ 00000 \times 10^{-1}$
256	2	$0.39062\ 50000\ 00000\ 00000 \times 10^{-2}$
4 096	3	$0.24414\ 06250\ 00000\ 00000 \times 10^{-3}$
65 536	4	$0.15258\ 78906\ 25000\ 00000 \times 10^{-4}$
1 048 576	5	$0.95367\ 43164\ 06250\ 00000 \times 10^{-6}$
16 777 216	6	$0.59604\ 64477\ 53906\ 25000 \times 10^{-7}$
268 435 456	7	$0.37252\ 90298\ 46191\ 40625 \times 10^{-8}$
4 294 967 296	8	$0.23283\ 06436\ 53869\ 62891 \times 10^{-9}$
68 719 476 736	9	$0.14551\ 91522\ 83668\ 51807 \times 10^{-10}$
1 099 511 627 776	10	$0.90949\ 47017\ 72928\ 23792 \times 10^{-12}$
17 592 186 044 416	11	$0.56843\ 41886\ 08080\ 14870 \times 10^{-13}$
281 474 976 710 656	12	$0.35527\ 13678\ 80050\ 09294 \times 10^{-14}$
4 503 599 627 370 496	13	$0.22204\ 46049\ 25031\ 30808 \times 10^{-15}$
72 057 594 037 927 936	14	$0.13877\ 78780\ 78144\ 56755 \times 10^{-16}$
1 152 921 504 606 846 976	15	$0.86736\ 17379\ 88403\ 54721 \times 10^{-18}$

POTENZEN DER ZAHL 10
(Dargestellt als Hexadezimalwerte)

10^n	n	10^{-n}	
1	0	1.0000 0000 0000 0000	
A	1	0.1999 9999 9999 999A	
64	2	0.28F5 C28F 5C28 F5C3	$\times 16^{-1}$
3E8	3	0.4189 374B C6A7 EF9E	$\times 16^{-2}$
2710	4	0.68DB 8BAC 710C B296	$\times 16^{-3}$
1 86A0	5	0.A7C5 AC47 1B47 8423	$\times 16^{-4}$
F 4240	6	0.10C6 F7A0 B5ED 8D37	$\times 16^{-4}$
98 9680	7	0.1AD7 F29A BCAF 4858	$\times 16^{-5}$
5F5 E100	8	0.2AF3 1DC4 6118 73BF	$\times 16^{-6}$
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC	$\times 16^{-7}$
2 540B E400	10	0.6DF3 7F67 5EF6 EADF	$\times 16^{-8}$
17 4876 E800	11	0.AFEB FF0B CB24 AAFF	$\times 16^{-9}$
E8 D4A5 1000	12	0.1197 9981 2DEA 1119	$\times 16^{-9}$
918 4E72 A000	13	0.1C25 C268 4976 81C2	$\times 16^{-10}$
5AF3 107A 4000	14	0.2D09 370D 4257 3604	$\times 16^{-11}$
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D	$\times 16^{-12}$
23 8652 6FC1 0000	16	0.734A CA5F 6226 F0AE	$\times 16^{-13}$
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449	$\times 16^{-14}$
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1	$\times 16^{-14}$
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35	$\times 16^{-15}$

Anhang D

Abweichung bei Computern mit Version 2 ROMS

Dieser Anhang beschreibt Unterschiede zwischen Computern mit ROM-Speichern der Version 3, für die die Kapitel dieses Buches geschrieben wurden, und Computern mit ROM-Speichern der Version 2.

KAPITEL 1: INBETRIEBNAHME

Beim Einschalten eines CBM-Computers mit Version-2-ROM-Speichern erscheinen Sternsymbole (*) anstelle der Nummernzeichen (#) in der ersten Bildschirmmeldung:

```
*** COMMODORE BASIC ***          VERSION 2 ROM-SPEICHER

### COMMODORE BASIC ###          VERSION 3 ROM-SPEICHER
```

Sie können an diesen Symbolen erkennen, mit welcher ROM-Version Ihr CBM-Computer ausgerüstet ist.

KAPITEL 4: TABELLEN (FELDER)

In CBM-Computern mit Version-2-ROMs ist die Gesamtzahl der Tabelleneintragungen (Feldelemente) für jede Tabelle auf 256 beschränkt. Ein Beispiel: Eine ein-spaltige Tabelle kann die Eintragungen 0 bis 255 enthalten. Eine zweispaltige Tabelle kann 0 bis 127 Eintragungen je Spalte enthalten: (0,0), (0,1) bis (0,127) und (1,0), (1,1) bis (1,127).

KAPITEL 5: KURSOR-PROGRAMMIERUNG

Bei Computern mit Version-2-ROMs wird der Kursor durch Ansprechen der Speicherstelle 548 sichtbar gemacht:

```
SICHTBARMACHEN DES KURSORS:

     80 POKE 548,0          VERSION 2 ROM-SPEICHER

     80 POKE 167,0          VERSION 3 ROM-SPEICHER
```

KAPITEL 5: FUNKTION RND

Die Funktion RND (0) hat keinen Funktionscharakter: Das Argument 0 liefert einen konstanten Funktionswert, der lediglich unter verschiedenen CBM-Computern verschieden sein kann.

Zur Erzeugung zufälliger Startzahlen verwenden Sie als Argument –TI. Dieses Verfahren wurde in allen Programmbeispielen für Zufallszahlen in Kapitel 5 verwendet. Das in Kapitel 5 beschriebene Programm für zufallsgestreute Bildschirmausgaben kann nicht auf Computern mit Version-2-ROMs ablaufen, da die verwendete Tabellengröße von 1000 Eintragungen die bei Version-2-ROMs bestehende Beschränkung auf 256 Elemente überschreitet. Eine programmtechnische Lösung besteht in der Bildung von 4 Tabellen zu je 250 Eintragungen.

KAPITEL 6: DATEIEN

Dieser Abschnitt gilt CBM-Benutzern, die beim Lesen von Dateien auf Bandkassette Schwierigkeiten haben: Wenn Sie einen CBM-Computer mit Version-2-ROMs benutzen und häufig Daten auf Dateien abzuspeichern haben, dann sollten Sie ernsthaft den Austausch von Version-2-ROMs gegen Version-3-ROMs erwägen.

Wenn Sie die Verwendung der Version-2-ROMs vorhaben, ist ein wenig mehr zusätzliche Programmierung erforderlich, um diese Probleme zu überwinden. Beim Schreiben von Daten auf Bandkassette werden bei Version-2-ROMs Zeiger nicht auf die Anfangsadresse des Pufferspeichers für Bandgeräte gesetzt und zwischen den Aufzeichnungen auf Band nicht genügend physikalischer Zwischenraum gelassen. Beim Lesen der Daten von der Bandkassette zeigt sich dann, daß Daten verloren gegangen oder verstümmelt worden sind. Hier einige Vorsichtsmaßnahmen, um diese Hindernisse zu überwinden.

1. ZEIGER AUF DIE ANFANGSADRESSE DES PUFFERSPEICHERS FÜR BANDGERÄTE SETZEN

Da CBM-Computer mit Version-2-ROMs die Initiierung der Startadresse im Pufferspeicher unterlassen, ehe eine logische Datei mit OPEN geöffnet wird, müssen Sie diese Initiierungen mit POKE-Anweisungen sicherstellen.

```
BANDGERAET #1:    POKE 243,122 : POKE 244,2 : OPEN 1,1,1
BANDGERAET #2:    POKE 243,58  : POKE 244,3 : OPEN 2,2,1
```

Die Zeiger zu den Anfangsadressen des verwendeten Pufferspeichers für Bandgeräte sind in den Speicheradressen 243 und 244 enthalten. Durch Speichern der oben genannten Zahlwerte mit POKE werden die Zeiger richtig initiiert.

2. VERGRÖSSERN DES PHYSIKALISCHEN ABSTANDS ZWISCHEN BANDAUFZEICHNUNGEN

Bei Bandaufzeichnungen mit CBM-Computern, die Version-2-ROMs verwenden, entsteht zwischen den Aufzeichnungen ein zu geringer physikalischer Abstand auf dem Band, so daß beim Lesen der Daten mit INPUT# oder GET# Lesefehler und Datenverluste entstehen können. Die folgende Programmroutine löst nach jedem Auslesen eines vollen Pufferspeichers auf die Bandkassette einen kurzen Bandvorlauf aus, wodurch der gewünschte physikalische Abstand zwischen Bandaufzeichnungen entsteht.

Vor Auslösen einer Aufzeichnungslücke muß festgestellt werden, ob der Pufferspeicher eine "physikalische Aufzeichnung" (oder einen "Datenblock") auf das Band geschrieben hat; ein voller Pufferspeicher ist Signal dafür, daß ein Datenblock gerade auf das Band geschrieben wurde, da der Inhalt des Pufferspeichers nur dann ausgelesen wird, wenn er bis zu seiner vollen Kapazität gefüllt wurde. In der Programmroutine wird daher zunächst festgestellt, wann der Pufferspeicher gefüllt ist, und dann ein kurzer Bandvorlauf ausgelöst.

2.1 ERKENNEN, WANN EIN PUFFERSPEICHER GEFÜLLT IST

Beim Schreiben von Daten auf Band wird nach jeder PRINT#-Anweisung die Länge jedes Datums berechnet und in einem Akkumulator-Speicher die Summe aller Datenlängen gebildet. Der Akkumulator-Inhalt wird mit der Speicherkapazität des Pufferspeichers von 191 Zeichen (Bytes) verglichen und bei Gleichheit der Auslesevorgang auf das Band solange unterbrochen, bis durch Bandvorlauf eine physikalische Aufzeichnungslücke hergestellt worden ist. Im folgenden Programmbeispiel sollen die Zahlen 1 bis 100 auf Band gespeichert werden:

```
10 POKE 243,122:POKE 244,2:OPEN 1,1,1  :REM ZEIGER INITIIEREN
20 FOR X=1 TO 100              :REM 100 ZAHLEN ERZEUGEN
30 PRINT#1,X                   :REM JEDE ZAHL X AUF BAND SPEICHERN
40 A=LEN(STR$(X))+1            :REM ANZAHL DER ZEICHEN IN X FESTST.
50 IF (QT+A)>=191 GOSUB 1000   :REM *SPRUNG INS UNTERPROGRAMM*
60 QT=QT+A                     :REM *  FUER BANDVORLAUF BEI  *
70 NEXT X                      :REM * VOLLEM  PUFFERSPEICHER *
80 CLOSE 1
```

Zeile 30 schreibt die Daten auf Band. Zur Feststellung der Länge einer Zeichenkette mit der Funktion LEN muß in diesem Fall die numerische Variable X in eine Zeichenkette verwandelt werden, wie Zeile 40 zeigt:

```
40 A=LEN(STR$(X))+1
```

Die Länge der Zeichenkette wird um 1 erhöht, um das Zeichen für Wagenrücklauf zu berücksichtigen, das nach jedem Datum auf Band geschrieben wird. Zeile 50 akkumuliert die Zahl der Zeichen in den vorangegangenen Zeichenketten, addiert die Länge der vorliegenden Zeichenkette und vergleicht die Summe mit der Speicherkapazität von 191 (Bytes). Wenn die Summe (QT + A) anzeigt, daß der Pufferspeicher voll ist, erfolgt mit GOSUB der Sprung in ein Unterprogramm zum Auslösen des Bandvorlaufs (siehe folgenden Abschnitt). Ist die Summe (QT + A) $<$ 191, dann wird in Zeile 60 der Akkumulator QT um A erhöht und der Schleifendurchlauf fortgesetzt, bis der Pufferspeicher gefüllt ist.

2.2 AUSLÖSEN DES BANDVORLAUFS

Zum Auslösen eines Bandvorlaufs sind folgende drei Schritte erforderlich:

1. Einschalten des Motors des Bandgeräts: POKE 59411,53
2. Beginn einer Programm-Warteschleife für die Dauer des Bandvorlaufs.
3. Ausschalten des Motors des Bandgeräts: POKE 59411,61.

Durch Speichern der Zahlenwerte 53 bzw. 61 in die Speicheradresse 59411 kann der Motor des Bandgeräts ein- bzw. ausgeschaltet werden. Sobald der Motor eingeschaltet wurde, läuft eine Warteschleife an, die mit den Zeittakten des Zeittakt-Gebers im CBM-Computer arbeitet. Hier das Beispiel für eine Warteschleife zwischen den POKE-Anweisungen:

```
1000 POKE 59411,53            :REM ***START DES BANDMOTORS ***
1010 T=TI
1020 IF (TI-T)<10 GOTO 1020 :REM 10 ZEITTAKTE ABWARTEN
1030 POKE 59411,61            :REM *** STOP DES BANDMOTORS***
1040 QT=0
1050 RETURN

RUN
```

Die eigentliche Warteschleife wird mit den Zeilen 1010 bis 1020 gebildet. Zeile 1010 setzt die Variable T auf den momentanen Wert des Zeittaktgebers TI. Der Wert von TI wird im Takt von 1/60stel Sekunde im 1 erhöht. Die Differenz (TI – T) stellt daher die seit Beginn der Warteschleife vergangene Zeit dar. Als Zeitvorgabe für den Bandvorlauf wurden 10 Zeittakte angenommen. Das Programm muß in Zeile 1020 solange warten, bis 10 Zeittakte abgelaufen sind. Danach wird in Zeile 1030 der Bandmotor ausgeschaltet.

Die Programmroutine berechnet den physikalischen Abstand zwischen Bandaufzeichnungen durch Vorgabe einer Laufzeit des Bandmotors von 10 Zeittakten. Diese Zeitvorgabe kann von Ihnen verändert werden, indem Sie in der Anweisung:

```
1020 IF (TI-T) < 10     GOTO 1020
```

den Wert 10 verändern; dieser Wert kann zwischen 5 und 30 schwanken.

In der Programmroutine ist folgendes Problem möglich: Wenn Ihr CBM-Computer für nahezu 24 Stunden in Betrieb ist, oder Sie die interne Uhr auf eine Zeit nahe 24:00:00 Uhr gesetzt haben, kann die Warteschleife aus folgendem Grund zu einer Falle werden: Wenn der Zeittaktzähler TI den Wert 5184000 (entsprechend 24:00:00 Uhr) erreicht hat, springt er zurück auf den Wert 0. Wurde T in Zeile 1010 noch kurz vor dem Sprung von TI einer der Werte um 5184000 zugewiesen, dann ist für 24 Stunden die Bedingung (TI – T) < 10 erfüllt. Hier ein Zahlenbeispiel: (0000008 – 5183998) < 10. Dieser Fall ist sehr unwahrscheinlich, sollte aber in Erinnerung behalten werden.

Eine andere Form der Warteschleife sieht wie folgt aus:

```
POKE 59411,53     :REM *** START DES BANDMOTORS ***
POKE 514,0        :REM ZEITTAKTGEBER ZURUECKSETZEN
WAIT 514,16       :REM 16 ZEITTAKTE ABWARTEN
POKE 59411,61     :REM ***  STOP DES BANDMOTORS ***
```

Mit der Anweisung POKE 514,0 wird der Wert 0 in das niederwertige Byte der internen Uhr gespeichert, wobei die vorhandene Zeittakt-Zählung gelöscht und die Uhr auf 0 gesetzt wird. Die Anweisung WAIT 514,16 verzögert den Programmablauf, bis 16 Zeittakte abgelaufen sind. Hat der Speicherinhalt der Adresse 514 den Wert 16 angenommen, dann schaltet die noch folgende POKE-Anweisung den Bandmotor aus.

Der Nachteil dieser Warteschleife ist, daß beim Rücksetzen des Zeittaktgebers auf 0 auch die im CBM-Computer gebildete Tageszeit verloren geht. Diese Programmroutine kann daher dort nicht verwendet werden, wo die genaue Uhrzeit im Programm benötigt wird.

Eine weitere Form der Warteschleife verwendet eine FOR-NEXT-Anweisung:

```
POKE 59411,53
FOR I=1 TO 60:NEXT I
POKE 59411,61
```

Mit der Zähldauer einer FOR-NEXT-Schleife läßt sich eine einfache, aber weniger genaue Warteschleife bilden. Der Vorteil jedoch ist, daß die oben genannten Probleme mit dem Zeittaktgeber entfallen.

2.3 DIE VOLLSTÄNDIGE PROGRAMMROUTINE

Das folgende Programmbeispiel verbindet die Programmroutine zum Erkennen, wann ein Pufferspeicher voll ist, mit der Programmroutine zum Auslösen eines befristeten Bandvorlaufs. Das Programm schreibt mit einer FOR-NEXT-Schleife 100 Zahlen auf Band. Nach jedem Schreibvorgang mit PRINT# wird geprüft, ob der Pufferspeicher gefüllt ist; ist das der Fall, dann erfolgt vor Ausführung der nächsten PRINT#-Anweisung ein Sprung in das Unterprogramm in Zeile 1000, mit dem ein auf 10 Zeittakte befristeter Bandvorlauf ausgelöst wird:

```
10 POKE 243,122 : POKE 244,2 : OPEN 1,1,1
20 FOR X=1 TO 100
30 PRINT#1,X
40 A=LEN(STR$(X))+1
50 IF (QT+A)>191 GOSUB 1000   :REM SPRUNG INS UNTERPROGRAMM
60 QT=QT+A                    :REM      FUER BANDVORLAUF BEI
70NEXT X                      :REM     VOLLEM PUFFERSPEICHER
80 CLOSE 1
90 END
1000 POKE 59411,53   :REM     *** START DES BANDMOTORS ***
1010 T=TI
1020 IF (TI-T)<10 GOTO 1020   :REM  10 ZEITTAKTE ABWARTEN
1030 POKE 59411,61   :REM      *** STOP DES BANDMOTORS ***
1040 QT=0            :REM     RUECKSETZEN DES AKKUMULATORS
1050 RETURN
```

mit **A** Länge der gedruckten Zeichenkette + 1 für Wagenrücklauf
QT Speicher zur Addition der Längen gedruckter Zeichenketten

Mit diesen Vorschlägen und Routinen sollten Sie keine Schwierigkeiten mehr beim Schreiben und Lesen von Daten auf Bandkassetten haben. Zeigen sich dennoch weiterhin Schwierigkeiten, dann sollten Sie Ihren CBM-Computer mit Version-3-ROM-Speichern ausrüsten.

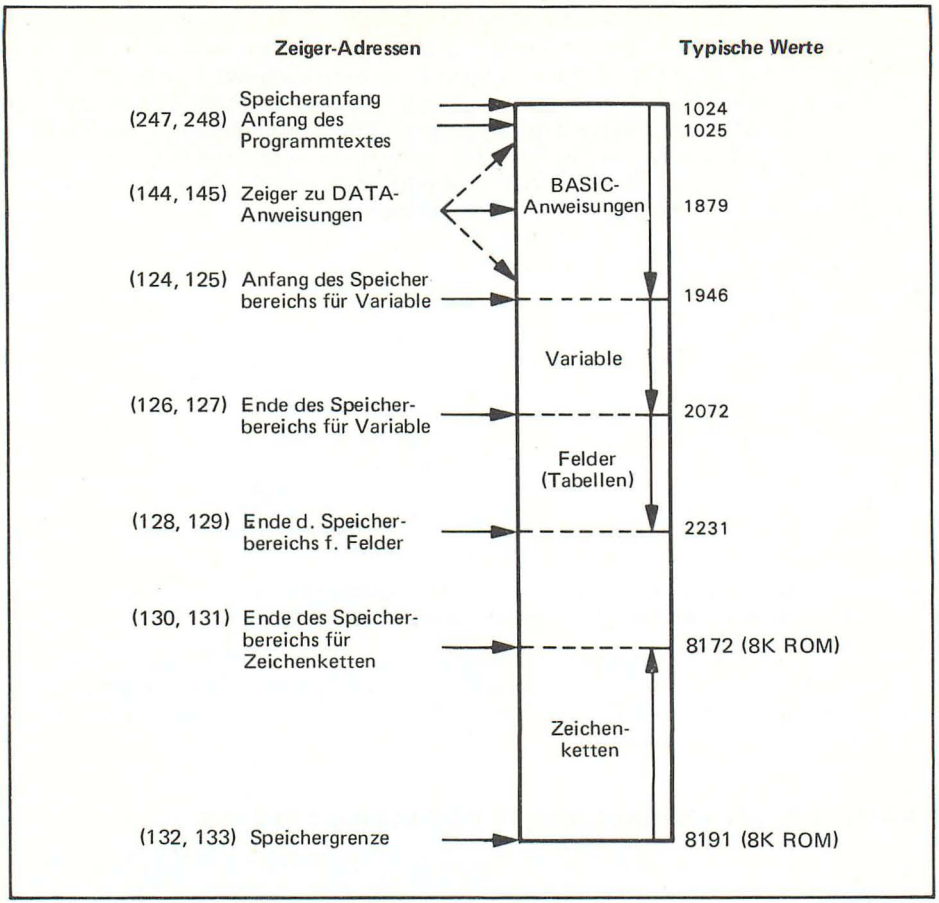
KAPITEL 7: SPEICHERAUSZÜGE IN TABELLEN D-1, D-2, D-3

Alle Änderungen in Kapitel 7 beruhen auf der Tatsache, daß die Speicherauszüge für Version-2-ROM-Speicher bei Entwicklung der Version-3-ROM-Speicher überarbeitet wurden.

Ausführliche Speicherauszüge für die verschiedenen Versionen von CBM BASIC sind in den Tabellen am Ende dieses Anhangs zu finden.

Die Tabellen D-1 und D-2 haben einen ähnlichen Aufbau; der Aufbau von Tabelle D-3 unterscheidet sich hiervon. In Tabellen D-1 und D-2 werden die Speicheradressen dezimal und hexadezimal angegeben; außerdem enthalten die Tabellen Beispiele für Speicherinhalte in Form dezimaler und hexadezimaler Werte. Tabelle D-3 enthält einen Vergleich der Speicherauszüge für BASIC 4.0 und BASIC 3.0 (das auch in Tabelle D-2 aufgeführt ist). Die Tabellenspalte DESCRIPTION enthält die gegenwärtig von Commodore verwendeten Beschreibungen der Speicherinhalte; die Tabellenspalte LABEL enthält die gegenwärtig von Commodore verwendeten Kennzeichnungen von Speicherstellen in Assembler-Sprache. In der Spalte BASIC 4.0 finden Sie die Adresse jeder Speicherstelle als Hexadezimalwert, während die Spalte BASIC 3.0 die entsprechende hexadezimale Adresse in BASIC 3.0 angibt. Um eine Speicherstelle in BASIC 4.0 zu finden, suchen Sie zuerst die hexadezimale Adresse in Tabelle D-2. Dann suchen Sie diese hexadezimale Adresse unter der Spalte BASIC 3.0 in Tabelle D-3 und lesen die gesuchte Adresse in der danebenstehenden Spalte BASIC 4.0 ab.

Mit Ausnahme der ersten beiden Eintragungen in Tabelle D-3, die die Speicheradresse 0000 darstellen, bezeichnen alle später auftretenden Adressen 0000 Ein-

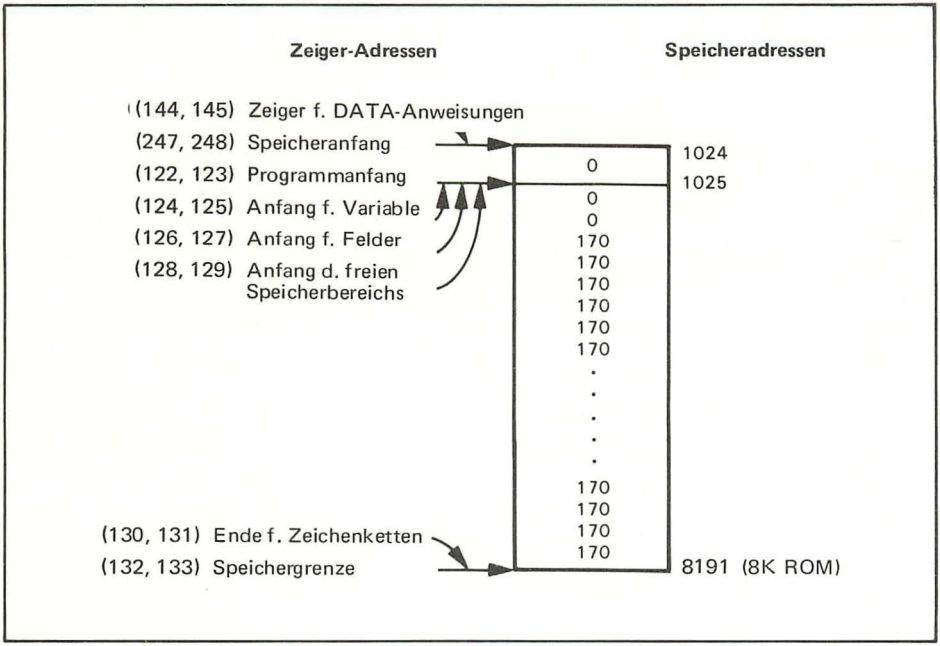


Figur D-1. Wichtigste Zeiger im Speicherbereich für Benutzerprogramme

tragungen, die in keiner der Versionen von BASIC existieren. Ein Beispiel: Wenn Sie in der Spalte BASIC 3.0 eine Adresse finden, deren zugehöriger Eintrag in der Spalte BASIC 4.0 den Wert 0000 hat, dann besitzt BASIC 4.0 keine entsprechende Speicherstelle in seinem Speicherauszug. Andererseits gibt eine Adresse 0000 in der Spalte BASIC 3.0 an, daß an dieser Stelle ein neuer Eintrag im Speicherauszug für BASIC 4.0 hinzugekommen ist, für den es in BASIC 3.0 keine Entsprechung gibt.

KAPITEL 7: CBM BASIC-ÜBERSETZER

In CBM-Computern mit Version-2-ROMs befinden sich die wichtigsten Zeiger in dem Speicherbereich für Benutzerprogramme an anderen Speicherstellen. Statt der Angaben in Kapitel 7, Figur 7-2 gelten die Angaben nach Figur D-1; entsprechend ersetzt Figur D-2 die Figur 7-4 in Kapitel 7.



Figur D-2. Zeiger bei Inbetriebnahme

KAPITEL 7: GLEITKOMMA-VARIABLE

In Kapitel 7 wurde bei der Besprechung des Datenformats, in dem Gleitkomma-Variable im Computer abgespeichert werden, ein Programm zur Untersuchung der Darstellung von Gleitkomma-Variablen angegeben. Bei Computern mit Version-2-ROMs muß Zeile 20 dieses Programms geändert werden:

```
10 INPUT A
20 X=PEEK(125)*256+PEEK(124)+2
30 PRINT A;"=";PEEK(X);PEEK(X+1);PEK(X+2);PEEK(X+3);PEEK(X+4)
40 GOTO 10
```

KAPITEL 7: ANFANGSZEIGER FÜR VARIABLE

Statt des Anfangszeigers (42, 43) zum Speicherbereich für Variable gilt bei Version-2-ROMs die Zeigeadresse (124, 125).

KAPITEL 7: TABELLEN (FELDER)

In Kapitel 7 wurde zur Untersuchung des Datenformats für die Speicherung von Tabellen ein Programm angegeben, das für CBM-Computer mit Version-2-ROMs in Zeilen 50 und 60 wie folgt zu ändern ist:

```
10 DIM A(5),B%(2,2),C$(10)          :REM DEMONSTRATIONS-TABELLEN (FELDER)
20 FOR I=0 TO 5 : A(I)=I : NEXT
30 FOR I=0 TO 2 : FOR J=0 TO 2 : B%(J,I)=100+3*I : NEXT J,I
40 FOR I=0 TO 10: C$(I)=CHR$(ASC("A")+I) : NEXT
50 X=PEEK(127)*256+PEEK(126)        :REM ANFANGSZEIGER ZU DEN FELDERN
60 Y=PEEK(129)*256+PEEK(128)        :REM ENDZEIGER ZU DEN FELDERN
70 FOR I=X TO Y
80 PRINT I,PEEK(I)
90 GET D$ : IF D$="" THEN GOTO 90    :REM TASTE DRUECKEN FUER
100 NEXT                             :REM      NAECHSTES ELEMENT
```

KAPITEL 7: PROGRAMMIEREN IN ASSEMBLER-SPRACHE

Im Abschnitt "Programmieren in Assembler-Sprache" in Kapitel 7 wurde besprochen, welche Möglichkeiten bestehen, Speicherbereiche für Assembler-Programme zu schaffen. Für CBM-Computer mit Version-2-ROMs ändert sich Abschnitt 2 in Kapitel 7 wie folgt:

2. Speicherbereich nahe der Speicherobergrenze. Die Speicherstellen 134 und 135 enthalten den Zeiger zur Speicherobergrenze. In CBM-Computern mit 8K RAM-Speichern ist die Adresse der Speicherobergrenze 8192. Sie können vorübergehend den Zeiger zur Speicherobergrenze auf eine niedrigere Adresse setzen, um eine Anzahl von Bytes zwischen dem neuen Zeigerwert und dem ursprünglichen Zeigerwert zur Speicherung von Assembler-Programmen zu reservieren. Um einen Speicherplatz von 1000 Bytes zu schaffen, müssen Sie den Wert des Zeigers auf 7192 (8192 - 1000) setzen, und diese Adresse in Byte-Form darstellen:

	Höherwertiges Byte		Niederwertiges Byte
$7192_{10} = 1C18_{16} \rightarrow 1C_{16} = 28_{10}$		und	$18_{16} = 24_{10}$

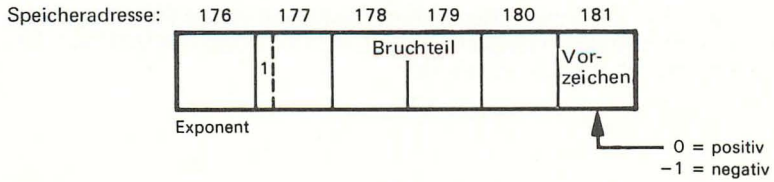
Mit den folgenden Anweisungen wird der Wert 24 in die Speicherstelle 134, und der Wert 28 in die Speicherstelle 135 gelesen:

```
10 AN=PEEK(134) : AH=PEEK(135) :REM ALTEN ZEIGER SPEICHERN
20 POKE 134,24  : POKE 135,28  :REM NEUE SPEICHERGRENZE=7192
 .
 .
 .
100 POKE 134,AN : POKE 135,AH  :REM ALTEN ZEIGER WIEDER EIN-
110 END                        :REM SETZEN
```

KAPITEL 7: USR

Da sich der Akkumulator bei CBM-Computern mit Version-2-ROMs an verschiedenen Speicherstellen befindet, muß die Beschreibung des Akkumulators in Kapitel 7 wie folgt geändert werden.

Die Übergabe von Parametern an Benutzerprogramme mit Hilfe der Funktion USR benutzt Speicherstellen, die als Akkumulator für Gleitkomma-Variable für alle Funktionen dient. Diese Speicherstellen werden von den 6 Bytes mit den Speicher adressen 176 bis 181 ($B0_{16} - B5_{16}$) gebildet. Die Speicherung erfolgt in folgendem Programm:



Wie bei der Speicherung von Gleitpunkt-Variablen wird eine normalisierte Exponentialdarstellung verwendet: Der wahre Exponent wird durch Addition des Zahlenwertes 128 in den Bereich 0 bis 255 verschoben und der Bruchteil derart normalisiert, daß das erste Bit nach dem Binärpunkt immer den Wert 1 hat. Der Unterschied zwischen diesem Format und dem Format für Variable besteht darin, daß die bei der Normalisierung entstandene 1 im höchstwertigen Bit von Byte 177 dargestellt wird. Ein weiteres Byte (181) dient zur Darstellung des Vorzeichens des Bruchteils. Dieses Format vereinfacht die Verarbeitung in Funktionen, die auf den Akkumulator zugreifen.

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
			Page 0 (0-255)	
				USR Function Locations
0	0000	76	4C	Constant 6502 JMP instruction
1-2	0001-0002	826	033A	User address jump vector
				Terminal I/O Maintenance
3	0003	0	00	Active input device number (0=keyboard)
4	0004	0	00	No. of nulls to print after CR/LF (0=normal)
5	0005	0	00	Cursor position for POS function (0-255)
6	0006	127	7F	Terminal width (unused)
7	0007	127	7F	Limit for scanning source columns (unused)
8	0008	60	3C	Line number storage preceding buffer
9	0009	3	03	Constant
10-89	000A-0059	48	30	BASIC input line buffer (80 bytes)
90	005A	0	00	General counter for BASIC
91	005B	0	00	Delimiter flag for quote mode scan
92	005C	255	FF	Input buffer pointer, general counter
				Evaluation of Variables
93	005D	0	00	Flag for dimensioned variables
94	005E	0	00	Flag for variable type: 00=numeric FF=string
95	005F	0	00	Flag for numeric variable type: 00=floating point 80=integer
96	0060	0	00	Flag to allow reserved words in strings and remarks
97	0061	0	00	Flag to allow subscripted variable
98	0062	0	00	Flag for input type: 0=INPUT 64=GET 152=READ
99	0063	0	00	Flag sign of TAN function
100	0064	0	00	Flag to suppress output: + normal -- suppressed
101	0065	104	68	Index to next available descriptor
102-103	0066-0067	101	0065	Pointer to last string temporary
104-111	0068-006F	2	0002	Table of double-byte descriptors that point to variables (8 bytes)
112-113	0070-0071	14525	38BD	Indirect index #1
114-115	0072-0073	62983	F607	Indirect index #2
116	0074	1	01	Pseudo-register for function operands (6 bytes)
117	0075	234	EA	
118	0076	0	00	
119	0077	0	00	
120	0078	0	00	
121	0079	0	00	

Tabelle D-1. Speicherauszug für CBM BASIC < 3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				Data BASIC Storage Maintenance
122-123	007A-007B	1025	0401	Pointer to start of text
124-125	007C-007D	1946	079A	Pointer to start of variables
126-127	007E-007F	2072	0818	Pointer to end of variables
128-129	0080-0081	2231	08B7	Pointer to end of arrays
130-131	0082-0083	8192	2000	Pointer to start of strings (moving down)
132-133	0084-0085	8191	1FFF	Pointer to end of strings (top of available RAM)
134-135	0086-0087	8192	2000	Pointer to limit of BASIC memory
136-137	0088-0089	2000	07D0	Line number of current line being executed −1 in 137=direct mode statement
138-139	008A-008B	110	006E	Line number for last line executed before CONT
140-141	008C-008D	1922	0782	Pointer to next line to be executed after CONT
142-143	008E-008F	1150	047E	Line number of current DATA line
144-145	0090-0091	1879	0757	Pointer to current DATA line
146-147	0092-0093	13	000D	Next DATA item within line
148-149	0094-0095	89	0059	Current variable name
150-151	0096-0097	2032	07F0	Pointer to current variable
152-153	0098-0099	2032	07F0	Pointer to next FOR. . . NEXT variable
154-155	009A-009B	31999	7CFF	Pointer to current operator in ROM table
156	009C	0	00	Mask for current logical operator
157-158	009D-009E	898	0382	Pointer to user function FN definition
159-160	009F-00A0	104	0068	Pointer to a string description
161	00A1	221	DD	Length of string
162	00A2	3	03	Constant used by garbage collection routine
163	00A3	76	4C	Constant 6502 JMP instruction
164-165	00A4-00A5	0	0000	Jump vector for user function FN
166-171	00A6-00AB	129	81	Floating point accumulator #3 (6 bytes)
172-173	00AC-00AD	0	00	Block transfer pointer #1
174-175	00AE-00AF	0	00	Block transfer pointer #2
176-181	00B0-00B5			Floating point accumulator (FAC)#1 (6 bytes)
		0	00	176 00B0 Exponent +128
		0	00	177 00B1 Fraction MSB Floating Point
		0	00	178 00B2 Fraction
		0	00	179 00B3 Fraction MSB Integer
		0	00	180 00B4 Fraction LSB
		0	00	181 00B5 Sign of fraction (0 if zero or positive, −1 if negative)
182	00B6	0	00	Copy of FAC #1 sign of fraction
183	00B7	0	00	Counter for number of bits to shift to normalize FAC #1
184-189	00B8-00BD	0	00	Floating point accumulator #2 (6 bytes)
190	00BE	0	00	Overflow byte for floating argument
191	00BF	0	00	Copy of FAC #2 sign of fraction
192-193	00C0-00C1	258	0102	Conversion pointer

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				RAM Subroutines
194-199	00C2-00C7	230	E6	Routine to fetch next BASIC character
200	00C8	173	AD	Entry to refetch current character
201-202	00C9-00CA	1929	0789	Pointer to source text
203-223	00CB-00DF	201	C9	Work area for RND function
				OS Page Zero Storage
224-225	00E0-00E1	33728	83C0	Pointer to start of line where cursor is flashing
226	00E2	0	00	Column position where cursor is flashing (0-79)
227-228	00E3-00E4	33792	8400	Utility pointer
229-230	00E5-00E6	1929	0789	End of current program
231-233	00E7-00E9	254	FE	Utility
234	00EA	0	00	Flag for quote mode. 0=not quote mode
235-237	00EB-00ED	192	C0	Utility
238	00EE	0	00	No. of characters in current file name
239	00EF	5	05	Current logical file number
240	00F0	255	FF	GPIB primary address
241	00F1	63	3F	GPIB device number
242	00F2	39	27	Max. no. of characters on current line (39,79)
243-244	00F3-00F4	634	027A	Pointer to start of current tape buffer (634 or 826)
245	00F5	23	17	Line number where cursor is flashing (0-24)
246	00F6	10	0A	I/O storage
247-248	00F7-00F8	1024	0400	OS pointer to program
249-250	00F9-00FA	3100	0C1C	Pointer to current file name
251	00FB	0	00	Number of Insert keys pushed to go
252	00FC	9	09	Serial bit shift word
253	00FD	0	00	Number of blocks remaining to read/write
254	00FE	0	09	Serial word buffer
255	00FF	243	F3	Overflow byte for binary to ASCII conversions
				Page 1 (256-511)
256-up	0100-up	32	20	Tape read working storage (up to 511) and conversion stg. 256-318 For error correction in tape reads (62 bytes) 256-266 Binary to ASCII conversion (11 bytes)
511-down	01FF-down	0	00	Stack (down to 256)
				Page 2-3 (512-1023)
				OS Working Storage
512-514	0200-0202	3801352	3A0108	24-hour clock incremented every 1/60 second (jiffy). Resets every 5,184,000 jiffies (24 hours). Stored in low to high order.

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
515	0203	255	FF	Matrix coordinate of key depressed at current jiffy. 1-80=key 255=no key
516	0204	0	00	Status of SHIFT key: 0=unshifted (up) 1=shifted (down)
517-518	0205-0206	37916	941C	Secondary jiffy clock
519	0207	52	34	Interrupt driver flag for cassette #1 ON switch
520	0208	0	00	Interrupt driver flag for cassette #2 ON switch
521	0209	255	FF	Keyswitch PIA
522	020A	0	00	Utility
523	020B	0	00	I/O flag 0=LOAD 1=VERIFY
524	020C	0	00	I/O status byte
525	020D	0	00	Number of characters in keyboard buffer (0 to 9)
526	020E	0	00	Flag to indicate reverse field on (0=normal)
527-536	020F-0218	85	55	Keyboard buffer (10 bytes)
537-538	0219-021A	34048	8500	Hardware interrupt vector
539-540	021B-021C	0	0000	6502 BRK instruction interrupt vector
541-546	021D-0222			Input routine storage (6 bytes)
		13	0D	542 021E No. of characters on screen line
547	0223	255	FF	Key image
548	0224	1	01	Flag for cursor enable: 0=Enable 1=Disable
549	0225	11	0B	Counter to flip cursor (20 to 1)
550	0226	32	20	Copy of character at current cursor position
551	0227	0	00	Flag for cursor on/off: 0=cursor moved 1=blink started
552	0228	0	00	Flag for tape write
553-577	0229-0241			High byte of screen line addresses 553-559=128 (lines 1-7) 560-565=129 (lines 8-13) 566-572=130 (lines 14-20) 573-577=131 (lines 21-25)
578-587	0242-024B	5	05	Table of logical numbers of open files
588-597	024C-0255	5	05	Table of device numbers of open files
598-607	0256-025F	255	FF	Table of secondary address modes of open files
608	0260	0	00	Flag for input source: 0=keyboard buffer 1=screen memory
609	0261	0	00	I/O utility
610	0262	1	01	Number of open files (index into tables)

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
611	0263	0	00	Default input device number (0=keyboard)
612	0264	3	03	Default output device number (3=screen)
613	0265	0	00	Tape parity byte
614	0266	0	00	I/O utility
615	0267	0	00	I/O utility
616	0268	0	00	Byte pointer in filename transfer
617	0269	0	00	I/O utility
618	026A	255	FF	I/O utility
619	026B	0	00	I/O utility
620	026C	8	08	Serial bit count
621	026D	0	00	Count of redundant tape blocks
622	026E	0	00	Tape utility
623	026F	0	00	Cycle counter flip for each bit read from tape
624	0270	0	00	Countdown synchronization on tape write
625	0271	0	00	Tape buffer 1 index to next character
626	0272	0	00	Tape buffer 2 index to next character
627	0273	0	00	Countdown synchronization on tape read
628	0274	0	00	Flag to indicate bit/byte tape error
629	0275	0	00	Flag to indicate tape error 0=first half-byte marker not written
630	0276	0	00	Flag to indicate tape error 0=2nd half-byte marker not written /Tape dropout counter
631	0277	0	00	Tape dropout counter
632	0278	128	80	Flag for tape read current function
633	0279	9	09	Checksum utility
634-825	027A-0339	1	01	Tape buffer for cassette #1 (192 bytes)
826-1017	033A-03F9	173	AD	Tape buffer for cassette #2 (192 bytes)
1018-1023	03FA-03FF	28	1C	Utility space/unused.
			Page 4-32 (1024-8191)	
1024-8191	0400-1FFF	0	00	User program area
			Page 33-128 (8192-32767)	
8192-32767	2000-7FFF	0	00	Expansion RAM
			Page 129-144 (32768-36863)	
32768-36863	8000-8FFF	12	0C	TV RAM 32768-33767 Display memory (1000 bytes)
			Page 145-192 (36864-49151)	
36864-49151	9000-BFFF	0	00	Expansion ROM
			Page 193-232 BASIC (49152-59391)	
				Pointers to BASIC Routines
49152-49153	C000-C001	50973	C71D	Pointer --1 to END*
49154-49155	C002-C003	50760	C648	Pointer --1 to FOR
49156-49157	C004-C005	52277	CC35	Pointer --1 to NEXT

These memory locations contain the address of the byte preceding the specified BASIC routines.

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
49158-49159	C006-C007	51183	C73F	Pointer -1 to DATA
49160-49161	C008-C009	51909	CAC5	Pointer -1 to INPUT#
49162-49163	C00A-C00B	51935	CADF	Pointer -1 to INPUT
49164-49165	C00C-C00D	53104	CF70	Pointer -1 to DIM
49166-49167	C00E-C00F	52003	CB23	Pointer -1 to READ
49168-49169	C010-C011	51356	C89C	Pointer -1 to LET
49170-49171	C012-C013	51100	C79C	Pointer -1 to GOTO
49172-49173	C014-C015	51060	C774	Pointer -1 to RUN
49174-49175	C016-C017	51231	C81F	Pointer -1 to IF
49176-49177	C018-C019	50956	C70C	Pointer -1 to RESTORE
49178-49179	C01A-C01B	51071	C77F	Pointer -1 to GOSUB
49180-49181	C01C-C01D	51145	C7C9	Pointer -1 to RETURN
49182-49183	C01E-C01F	51250	C832	Pointer -1 to REM
49184-49185	C020-C021	50971	C71B	Pointer -1 to STOP
49186-49187	C022-C023	51266	C842	Pointer -1 to ON
49188-49189	C024-C025	55041	D701	Pointer -1 to WAIT
49190-49191	C026-C027	65492	FFD4	Pointer -1 to LOAD
49192-49193	C028-C029	65495	FFD7	Pointer -1 to SAVE
49194-49195	C02A-C02B	65498	FFDA	Pointer -1 to VERIFY
49196-49197	C02C-C02D	53908	D294	Pointer -1 to DEF
49198-49199	C02E-C02F	55032	D6F8	Pointer -1 to POKE
49200-49201	C030-C031	51582	C97E	Pointer -1 to PRINT#
49202-49203	C032-C033	51614	C99E	Pointer -1 to PRINT
49204-49205	C034-C035	51012	C744	Pointer -1 to CONT
49206-49207	C036-C037	50599	C5A7	Pointer -1 to LIST
49208-49209	C038-C039	51055	C76F	Pointer -1 to CLR
49210-49211	C03A-C03B	51588	C984	Pointer -1 to CMD
49212-49213	C03C-C03D	65501	FFDD	Pointer -1 to SYS
49214-49215	C03E-C03F	65471	FFBF	Pointer -1 to OPEN
49216-49217	C040-C041	65474	FFC2	Pointer -1 to CLOSE
49218-49219	C042-C043	51870	CA9E	Pointer -1 to GET
49220-49221	C044-C045	50512	C550	Pointer -1 to NEW
49222-49223	C046-C047	56075	DB0B	Pointer to SGN**
49224-49225	C048-C049	56222	DB9E	Pointer to INT
49226-49227	C04A-C04B	56106	DB2A	Pointer to ABS
49228-49229	C04C-C04D	0	0000	Pointer to USR pointer
49230-49231	C04E-C04F	53860	D264	Pointer to FRE
49232-49233	C050-C051	53893	D285	Pointer to POS
49234-49235	C052-C053	56868	DE24	Pointer to SQR
40236-49237	C054-C055	57157	DF45	Pointer to RND
49238-49239	C056-C057	55487	D8BF	Pointer to LOG
49240-49241	C058-C059	56992	DEA0	Pointer to EXP
49242-49243	C05A-C05B	57246	DF9E	Pointer to COS
49244-49245	C05C-C05D	57253	DFA5	Pointer to SIN
49246-49247	C05E-C05F	57326	DFEE	Pointer to TAN
49248-49249	C060-C061	57416	E048	Pointer to ATN
49250-49251	C062-C063	55014	D6E6	Pointer to PEEK
49252-49253	C064-C065	54868	D654	Pointer to LEN
49254-49255	C066-C067	54089	D349	Pointer to STR$
49256-49257	C068-C069	54917	D685	Pointer to VAL
49258-49259	C06A-C06B	54883	D663	Pointer to ASC
49260-49261	C06C-C06D	54724	D5C4	Pointer to CHR$
49262-49263	C06E-C06F	54744	D5D8	Pointer to LEFT$

** These memory locations contain the address of the first byte of the specified BASIC routines.

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
49158-49159	C006-C007	51183	C73F	Pointer -1 to DATA
49160-49161	C008-C009	51909	CAC5	Pointer -1 to INPUT#
49162-49163	C00A-C00B	51935	CADF	Pointer -1 to INPUT
49164-49165	C00C-C00D	53104	CF70	Pointer -1 to DIM
49166-49167	C00E-C00F	52003	CB23	Pointer -1 to READ
49168-49169	C010-C011	51356	C89C	Pointer -1 to LET
49170-49171	C012-C013	51100	C79C	Pointer -1 to GOTO
49172-49173	C014-C015	51060	C774	Pointer -1 to RUN
49174-49175	C016-C017	51231	C81F	Pointer -1 to IF
49176-49177	C018-C019	50956	C70C	Pointer -1 to RESTORE
49178-49179	C01A-C01B	51071	C77F	Pointer -1 to GOSUB
49180-49181	C01C-C01D	51145	C7C9	Pointer -1 to RETURN
49182-49183	C01E-C01F	51250	C832	Pointer -1 to REM
49184-49185	C020-C021	50971	C71B	Pointer -1 to STOP
49186-49187	C022-C023	51266	C842	Pointer -1 to ON
49188-49189	C024-C025	55041	D701	Pointer -1 to WAIT
49190-49191	C026-C027	65492	FFD4	Pointer -1 to LOAD
49192-49193	C028-C029	65495	FFD7	Pointer -1 to SAVE
49194-49195	C02A-C02B	65498	FFDA	Pointer -1 to VERIFY
49196-49197	C02C-C02D	53908	D294	Pointer -1 to DEF
49198-49199	C02E-C02F	55032	D6F8	Pointer -1 to POKE
49200-49201	C030-C031	51582	C97E	Pointer -1 to PRINT#
49202-49203	C032-C033	51614	C99E	Pointer -1 to PRINT
49204-49205	C034-C035	51012	C744	Pointer -1 to CONT
49206-49207	C036-C037	50599	C5A7	Pointer -1 to LIST
49208-49209	C038-C039	51055	C76F	Pointer -1 to CLR
49210-49211	C03A-C03B	51588	C984	Pointer -1 to CMD
49212-49213	C03C-C03D	65501	FFDD	Pointer -1 to SYS
49214-49215	C03E-C03F	65471	FFBF	Pointer -1 to OPEN
49216-49217	C040-C041	65474	FFC2	Pointer -1 to CLOSE
49218-49219	C042-C043	51870	CA9E	Pointer -1 to GET
49220-49221	C044-C045	50512	C550	Pointer -1 to NEW
49222-49223	C046-C047	56075	DB0B	Pointer to SGN**
49224-49225	C048-C049	56222	DB9E	Pointer to INT
49226-49227	C04A-C04B	56106	DB2A	Pointer to ABS
49228-49229	C04C-C04D	0	0000	Pointer to USR pointer
49230-49231	C04E-C04F	53860	D264	Pointer to FRE
49232-49233	C050-C051	53893	D285	Pointer to POS
49234-49235	C052-C053	56868	DE24	Pointer to SQR
40236-49237	C054-C055	57157	DF45	Pointer to RND
49238-49239	C056-C057	55487	D8BF	Pointer to LOG
49240-49241	C058-C059	56992	DEA0	Pointer to EXP
49242-49243	C05A-C05B	57246	DF9E	Pointer to COS
49244-49245	C05C-C05D	57253	DFA5	Pointer to SIN
49246-49247	C05E-C05F	57326	DFEE	Pointer to TAN
49248-49249	C060-C061	57416	E048	Pointer to ATN
49250-49251	C062-C063	55014	D6E6	Pointer to PEEK
49252-49253	C064-C065	54868	D654	Pointer to LEN
49254-49255	C066-C067	54089	D349	Pointer to STR$
49256-49257	C068-C069	54917	D685	Pointer to VAL
49258-49259	C06A-C06B	54883	D663	Pointer to ASC
49260-49261	C06C-C06D	54724	D5C4	Pointer to CHR$
49262-49263	C06E-C06F	54744	D5D8	Pointer to LEFT$

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				51910 CAC6 INPUT#
				51936 CAED INPUT
				51991 CB17 Input prompt
				52004 CB24 READ
				52242 CC12 Error messages
				52278 CC36 NEXT
				52370 CC92 Format checker
				52408 CCB8 Expression evaluator
				52538 CD3A Stack argument
				52637 CD9D Symbol evaluator
				52668 CDBC Pi
				53105 CF71 DIM
				53207 CFD7 Variable table look-up
				53415 D0A7 Floating-to-integer
				53860 D264 FRE
				53880 D278 Integer-to-floating
				53893 D285 POS
				53909 D295 DEF
				54089 D349 STR$
				54724 D5C4 CHR$
				54744 D5D8 LEFT$
				54788 D604 RIGHT$
				54799 D60F MID$
				54868 D654 LEN
				54883 D663 ASC
				54917 D685 VAL
				55014 D6E6 PEEK
				55033 D6F9 POKE
				55042 D702 WAIT
				55080 D728 Subtraction
				55103 D73F Addition
				55487 D8BF LOG
				55552 D900 Multiplication
				55646 D95E Load number to AFAC
				55650 D962 Load variable to AFAC
				55780 D9E4 Division
				55924 DA74 Load Accumulator (FAC)
				55928 DA78 Load variable to FAC
				55979 DAAB Store variable from FAC
				56075 DB0B SGN
				56106 DB2A ABS
				56222 DB9E INT
				56868 DE24 SQR
				56878 DE2E Raise AFAC to power FAC
				56992 DEA0 EXP
				57157 DF45 RND
				57246 DF9E COS
				57253 DFA5 SIN
				57326 DFEE TAN

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
57344-59391	E000-E7FF			**Screen Editor** Starting Address Function 57416 E048 ATN 57525 E0B5 Initialize BASIC system 57910 E236 Clear screen 57981 E27D Character fetch
58004-58986	E294-E66A			Video driver 58282 E3AA Scroll processor 58346 E3EA Video display routine 58185 E349 Quote mode ($EA) switcher 58346 E3EA Print character 58713 E559 Scroll 1 line 58758 E586 Interrupt Request (IRQ)
58987-59012	E66B-E684			Interrupt handler
59013-59198	E685-E73E			Clock update
59199-59227	E73F-E75B			Keyboard scan
59228-59348	E75C-E7D4			Keyboard encoding table
	Page 233-240 I/O Ports and Expansion I/O (PIA's and VIA) (59392-61439)			
				Keyboard PIA (59408-59411)
59408	E810	233	E9	I/O Port A and Data Direction register
59409	E811	60	3C	Control Register A — screen blanking 52=Screen off (blanked) 60=Screen on
59410	E812	255	FF	I/O Port B and Data Direction register 255=all keys except: 254=RVS key 253=key 251=SPACE key 247= < key
59411	E813	61	3D	Control Register B — #1 cassette motor 53=motor on 61=motor off
				IEEE Port PIA (59424-59427)
59424	E820	255	FF	I/O Port A and Data Direction register PEEK (59424) reads input data.
59425	E821	188	BC	Control Register A — set output line CA2 POKE 59425,52=low POKE 59425,60=high
59426	E822	255	FF	I/O Port B and Data Direction register POKE 59426,data writes output data POKE 59426,255 before a read to Port A
59427	E823	60	3C	Control Register B — set output line CB2 POKE 59427,52=low POKE 59427,60=high

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				Parallel User Port VIA (59456-59471)
59456	E840	254	FE	I/O Port B 207=#2 cassette motor on 223=#2 cassette motor off WAIT 59456,23,23 waits for vertical retrace of display Bit 1=PB1 (NFRD on IEEE connector) output line Bit 3=PB3 (ATN on IEEE connector) output line
59457	E841	255	FF	I/O Port A with handshaking
59458	E842	30	1E	Data Direction register for I/O Port B
59459	E843	0	00	Data Direction register for I/O Port A For each bit 1=output, 0=input =0 all input =255 all output
59460-59461	E844-E845	25248	62A0	(Low, high order) Read Timer 1 Counter; write to Timer 1 Latch and (high byte) initiate count
59462-59463	E846-E847	65381	FF65	(Low, high order) Read Timer 1 Latch
59464	E848	113	71	Read Timer 2 Counter low byte and reset interrupt; write to Timer 2 low byte PEEK (59464) Clock decrements every microsecond POKE 59464,n sets SR rate of shift from high (n=0) to low (n=255) for music from User Port.
59465	E849	200	C8	Read Timer 2 Counter high byte; write to Timer 2 high byte and reset interrupt. PEEK (59465) Clock decrements every 256 microseconds
59466	E84A	1	01	Serial I/O Shift register (SR) POKE 59466,15 or 51 or 85 to generate square wave output at CB2 for playing music from User Port.
59467	E84B	0	00	Auxiliary Control register. =16 Sets SR to free-running mode for music from User Port. =0 for proper operation of tape drive
59468	E84C	14	0E	Peripheral Control register =12 for graphics on shifted characters =14 for lower-case letters on shifted characters
59469	E84D	0	00	Interrupt Flag register
59470	E84E	128	80	Interrupt Enable register
59471	E84F	255	FF	I/O Port A without handshaking
		Page 241-256 Operating System (61440-65535)		
61622-61904	F0B6-F1D0			**File Control**
61905-63532	F1D1-F82C			**Starting Address** **Function** 61905 F1D1 Get a character (without cursor) 61921 F1E1 Input a character (with cursor)

Tabelle D-1. Speicherauszug für CBM BASIC <3.0, Version 2 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				62002 F232 Display a character
				62026 F24A Close all files
				62121 F2A9 CLOSE
				62250 F32A STOP search
				62278 F346 Tape playback
				62402 F3C2 LOAD
				62481 F411 Display filename
				62515 F433 Fetch file number
				62556 F45C Number fetch
				62647 F4B7 VERIFY
				62724 F504 Fetch filename
				62741 F515 Fetch tape character
				62753 F521 OPEN
				62824 F568 Record SAVE routine
				62894 F5AE Tape header search
				62947 F5E3 Clear current tape buffer
				62957 F5ED Write tape end block
				63101 F67D Set up tape end pointer
				63108 F684 SYS
				63134 F69E SAVE
				63153 F6B1 SAVE memory block on cassette
				63273 F729 Update secondary jiffy clock
63533-64789	F82D-FD15			**Tape Control**
				63582 F85E Check for cassette on
				63615 F87F Tape read to buffer
				63684 F8C4 Write block to tape
				63765 F915 Interrupt wait
64824-65458	FD38-FFB2			**Power-On Diagnostics**
				64824 FD38 System reset SYS (64824) simulates power-on reset
				64909 FD8D Reset BASIC (does not affect User Program)
				64912 FD90 EOT-buffer compare
65472-65516	FFC0-FFEC			**Jump Vectors**
65472-65474	FFC0-FFC2	76 62753	4C F521	JMP OPEN
65475-65477	FFC3-FFC5	76 62121	4C F2A9	JMP CLOSE
65487-65489	FFCF-FFD1	76 61921	4C F1E1	JMP RDT
65490-65492	FFD2-FFD4	76 62002	4C F232	JMP WRT
65493-65495	FFD5-FFD7	76 62402	4C F3C2	JMP LOAD
65496-65498	FFD8-FFDA	76 63134	4C F69E	JMP SAVE
65499-65501	FFDB-FFDD	76 62647	4C F4B7	JMP VERIFY
65502-65504	FFDE-FFED	76 63108	4C F684	JMP SYS
65508-65510	FFE4-FFE6	76 61905	4C F1D1	JMP GETC
65514-65516	FFEA-FFEC	76 63273	4C F729	JMP Clock Update
65530-65535	FFFA-FFFF			**6502 Interrupt Vectors**
65530-65531	FFFA-FFFB	51808	CA60	Non-maskable interrupt (NMI)
65532-65533	FFFC-FFFD	64824	FD38	System reset (RESET)
65534-65535	FFFE-FFFF	58987	E66B	Interrupt request, break (IRQ+BRK)

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
			Page 0 (0-255)	
				USR Function Locations
0	0000	76	4C	Constant 6502 JMP instruction
1-2	0001-0002	826	033A	User address jump vector
				Evaluation of Variables and Terminal I/O Maintenance
3	0003	0	00	Search character
4	0004	0	00	Delimiter flag for quote mode scan
5	0005	255	FF	Input buffer pointer, general counter
6	0006	0	00	Flag for dimensioned variables
7	0007	0	00	Flag for variable type: 00=numeric FF=string
8	0008	0	00	Flag for numeric variable type: 00=floating point 80=integer
9	0009	0	00	Flag for DATA scan; LIST quote; memory
10	000A	0	00	Flag to allow subscripted variable; FNx flag
11	000B	0	00	Flag for input type: 0=INPUT 64=GET 152=READ
12	000C	0	00	Flag for ATN sign; comparison evaluation
13	000D	0	00	Flag to suppress output: + normal -- suppressed
14	000E	0	00	Current I/O device for prompt-suppress
15	000F	40	28	Terminal width (unused)
16	0010	30	1E	Limit for scanning source columns (unused)
17-18	0011-0012	828	033C	Basic integer address (for SYS, GOTO, etc.)
19	0013	22	16	Index to next available descriptor
20-21	0014-0015	19	13	Pointer to last string temporary
22-29	0016-001D	2	0002	Table of double-byte descriptions that point to variables (8 bytes)
30-31	001E-001F	16451	4043	Indirect index #1
32-33	0020-0021	26119	6607	Indirect index #2
34	0022	1	01	Pseudo-register for function operands (6 bytes)
35	0023	140	8C	
36	0024	0	00	
37	0025	0	00	
38	0026	0	00	
39	0027	0	00	

Tabelle D-2. Speicherauszug für CBM BASIC H3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				Data Storage Maintenance
40-41	0028-0029	1025	0401	Pointer to start of BASIC text
42-43	002A-002B	1920	0780	Pointer to start of variables
44-45	002C-002D	2032	07F0	Pointer to end of variables
46-47	002E-002F	2191	088F	Pointer to end of arrays
48-49	0030-0031	8192	2000	Pointer to start of strings (moving down)
50-51	0032-0033	8191	1FFF	Pointer to end of strings (top of available RAM)
52-53	0034-0035	8192	2000	Pointer to limit of BASIC memory
54-55	0036-0037	2000	07D0	Current line number. Loc. 55=2 if no program yet executed
56-57	0038-0039	110	006E	Previous line number
58-59	003A-003B	1897	0769	Pointer to next line to be executed (for CONT)
60-61	003C-003D	200	00C8	Line number of current DATA line
62-63	003E-003F	1855	073F	Pointer to current DATA item
				Expression Evaluation
64-65	0040-0041	514	0202	INPUT vector
66-67	0042-0043	89	0059	Current variable name
68-69	0044-0045	2006	07D6	Pointer to current variable
70-71	0046-0047	2006	07D6	Pointer to current FOR . . . NEXT variable
72-73	0048-0049	1279	04FF	Pointer to current operator in ROM table
74	004A	0	00	Mask for current logical operator
75-76	004B-004C	62268	F33C	Pointer to user function FN definition
77-78	0040-004E	26531	67A3	Pointer to a string description
79	004F	243	F3	Length of string
80	0050	3	03	Constant used by garbage collection routine
81	0051	76	4C	Constant 6502 JMP instruction
82-83	0052-0053	0	00	Jump vector for functions
84-89	0054-0059	211	D3	Floating point accumulator #3 (6 bytes)
90-91	005A-005B	0	0000	Block transfer pointer #1
92-93	005C-005D	0	0000	Block transfer pointer #2
94-99	005E-0063			Floating point accumulator (FAC) #1 (6 bytes)
		0	00	94 005E Exponent +128
		0	00	95 005F Fraction MSB Floating Point
		0	00	96 0060 Fraction
		0	00	97 0061 Fraction MSB Integer
		0	00	98 0062 Fraction LSB
		0	00	99 0063 Sign of fraction (0 if zero or positive, −1 if negative)
100	0064	0	00	Copy of FAC #1 sign of fraction
101	0065	0	00	Counter for number of bits to shift to normalize FAC #1
102-107	0066-006B	0	00	Floating point accumulator #2 (6 bytes)
108	006C	0	00	Overflow byte for floating argument
109	006D	0	00	Copy of FAC #2 sign of fraction
110-111	006E-006F	258	0102	Conversion pointer

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				RAM Subroutines
112-135	0070-0087	230	E6	Routine to fetch next BASIC character
		173	AD	118 76 Entry to refetch current character
		1904	0770	119-120 77-78 Pointer into source text
136-140	0088-008C	128	80	Next random no. in storage and RND work area
				OS Page Zero Storage
141-143	008D-008F	398710	061576	24-hour clock incremented every 1/60 second (jiffy). Resets every 5,184,000 jiffies (24 hours). Stored in high to low order
144-145	0090-0091	58926	E62E	Hardware interrupt vector
146-147	0092-0093	64791	FD17	6502 BRK instruction interrupt vector
148-149	0094-0095	50057	C389	NMI interrupt vector
150	0096	0	00	Status word ST (1 byte)
151	0097	255	FF	Matrix coordinate of key depressed at current jiffy. 1-80=key. 255=no key
152	0098	0	00	Status of SHIFT key: 0=unshifted (up) 1=shifted (down)
153-154	0099-009A	65282	FF02	Correction factor for clock
155	009B	255	FF	Keyswitch PIA: STOP and RVS flags
156	009C	0	00	Timing constant buffer
157	009D	0	00	I/O flag: 0=LOAD 1=VERIFY
158	009E	0	00	Number of characters in keyboard buffer (0 to 9)
159	009F	0	00	Flag to indicate reverse field on (0=normal)
160	00A0	0	00	IEEE 488 output flag FF=character waiting
161	00A1	13	0D	Byte pointer to end of line for input
162	00A2	0	00	Utility
163-164	00A3-00A4	11, 13	0B, 0D	Cursor log (row, column)
165	00A5	63	3F	IEEE 488 output character buffer
166	00A6	255	FF	Key image
167	00A7	1	01	Flag for cursor enable: 0=Enable 1=Disable
168	00A8	17	11	Counter to flip cursor (20 to 1)
169	00A9	32	20	Copy of character at current cursor position
170	00AA	0	00	Flag for cursor on/off: 0=cursor moved 1=blink started
171	00AB	0	00	Flag for tape write
172	00AC	0	00	Flag for input source: 0=keyboard buffer 1=screen memory

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				OS Page Zero Storage (Continued)
173	00AD	0	00	I/O utility: X save flag
174	00AE	1	01	Number of open files (index into tables)
175	00AF	0	00	Default input device number (0=keyboard)
176	00B0	3	03	Default output device number (3=screen)
177	00B1	0	00	Tape parity byte
178	00B2	0	00	Flag for byte received
179	00B3	0	00	I/O utility
180	00B4	0	00	Tape buffer character
181	00B5	0	00	Byte pointer in filename transfer
182	00B6	0	00	I/O utility
183	00B7	0	00	Serial bit count
184	00B8	0	00	Tape utility
185	00B9	0	00	Cycle counter — flip for each bit read from tape
186	00BA	0	00	Countdown synchronization on tape write
187	00BB	0	00	Tape buffer 1 index to next character
188	00BC	0	00	Tape buffer 2 index to next character
189	00BD	0	00	Countdown synchronization on tape read
190	00BE	0	00	Flag to indicate bit/byte tape error
191	00BF	0	00	Flag to indicate tape error 0=first half-byte marker not written
192	00C0	0	00	Flag to indicate tape error 0=2nd half-byte marker not written
193	00C1	0	00	Tape dropout counter
194	00C2	0	00	Flag for cassette read current function 0=scan, 1-15=count, 40_{16}=load, 80_{16}=end
195	00C3	0	00	Checksum utility
196-197	00C4-00C5	33728	83CD	Pointer to start of line where cursor is flashing
198	00C6	0	00	Column position where cursor is flashing (0-79)
199-200	00C7-00C8	33792	8400	Load start address; utility pointer
201-202	00C9-00CA	0	0000	Load end address
203-204	00CB-00CC	0	00	Tape timing constants
205	00CD	0	00	Flag for quote mode 0=not quote mode
206	00CE	0	00	Flag for tape read timer enable 0=disabled
207	00CF	0	00	Flag for EOT received from tape
208	00D0	0	00	Read character error
209	00D1	0	00	No. of characters in current file name
210	00D2	4	04	Current logical file number
211	00D3	255	FF	Current secondary address
212	00D4	4	04	Current device number
213	00D5	39	27	Current screen line length (39, 79)
214-215	00D6-00D7	0	0000	Pointer to start of current tape buffer (634 or 826)

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
216	00D8	24	18	Line number where cursor is flashing (0-24)
217	00D9	10	0A	I/O storage: last key input, buffer checksum, bit buffer
218-219	00DA-00DB	0	0000	Pointer to current file name
220	00DC	0	00	Number of Insert keys pushed to go
221	00DD	0	00	Serial bit shift word
222	00DE	0	00	Number of blocks remaining to read/write
223	00DF	0	00	Serial word buffer
224-248	00E0-00F8			High byte of screen line addresses
		128	80	224-230=128 (lines 1-7)
		129	81	231-236=129 (lines 8-13)
		130	82	237-243=130 (lines 14-20)
		131	83	244-248=131 (lines 21-25)
249	00F9	0	00	Cassette #1 status switch
250	00FA	0	00	Cassette #2 status switch
251-252	00FB-00FC	54144	D380	Tape start address
253-255	00FD-00FF	243	F3	Utility
				Page 1 (256-511)
256-up	0100-up	32	20	Tape read working storage (up to 511) and conversion storage 256-318 For error correction in tape reads (62 bytes) 256-266 Binary to ASCII conversion (11 bytes)
511-down	01FF-down	44	2C	Stack (down to 256)
				Page 2-3 (512-1023)
512-592	0200-0250			BASIC input line buffer (80 bytes)
		12597	3135	512-513 0200-0201 Program Counter
		50	32	514 0202 Processor status
		0	00	515 0203 Accumulator
		171	AB	516 0204 X index
		0	00	517 0205 Y index
		0	00	518 0206 Stack pointer
		15104	3B00	519-520 0207-0208 User modifiable IRQ
593-602	0251-025A	4	04	Table of logical numbers of open files
603-612	025B-0264	4	04	Table of device numbers of open files
613-622	0265-026E	255	FF	Table of secondary address modes of open files
623-632	026F-0278	3	03	Keyboard buffer (10 bytes)
633	0279	28	1C	Keyboard utility
634-825	027A-0339	28	1C	Tape buffer for cassette #1 (192 bytes)
826-1017	033A-03F9	173	AD	Tape buffer for cassette #2 (192 bytes)
1018-1019	03FA-03FB	59383	E7F7	Vector for Machine Language Monitor
1020-1023	03FC-03FF	195	C3	Utility space/unused

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				OS Page Zero Storage (Continued)
			Page 4-128 (1024-32767)	
1024-32767	0400-7FFF	0	00	User program area and Expansion RAM 4K PET: 1024-4095 0400-0FFF User program area 4096-32767 1000-7FFF Expansion RAM 8K PET: 1024-8191 0400-1FFF User program area 8192-32767 2000-7FFF Expansion RAM 16K PET: 1024-16383 0400-3FFF User program area 16384-32767 4000-7FFF Expansion RAM 32K PET: 1024-32767 0400-7FFF User program area
			Page 129-144 (32768-36863)	
32768-36863	8000-8FFF	32	20	TV RAM 32768-33767 Display memory (1000 bytes)
			Page 145-192 (36864-49151)	
36864-49151	9000-BFFF	144	90	Expansion ROM
			Page 193-232 BASIC (49152-59391)	
				Pointers to BASIC Routines
49152-49153	C000-C001	51008	C740	Pointer --1 to END*
49154-49155	C002-C003	50775	C657	Pointer --1 to FOR
49156-49157	C004-C005	52255	CC1F	Pointer --1 to NEXT
49158-49159	C006-C007	51199	C7FF	Pointer --1 to DATA
49160-49161	C008-C009	51878	CAA6	Pointer --1 to INPUT#
49162-49163	C00A-C00B	51904	CAC0	Pointer --1 to INPUT
49164-49165	C00C-C00D	53090	CF62	Pointer --1 to DIM
49166-49167	C00E-C00F	51974	CB06	Pointer --1 to READ
49168-49169	C010-C011	51372	C8AC	Pointer --1 to LET
49170-49171	C012-C013	51116	C7AC	Pointer --1 to GOTO
49172-49173	C014-C015	51076	C784	Pointer --1 to RUN
49174-49175	C016-C017	51247	C82F	Pointer --1 to IF
49176-49177	C018-C019	50991	C72F	Pointer --1 to RESTORE
49178-49179	C01A-C01B	51087	C78F	Pointer --1 to GOSUB
49180-49181	C01C-C01D	51161	C7D9	Pointer --1 to RETURN
49182-49183	C01E-C01F	51266	C842	Pointer --1 to REM
49184-49185	C020-C021	51006	C73E	Pointer --1 to STOP
49186-49187	C022-C023	51282	C852	Pointer --1 to ON
49188-49189	C024-C025	55055	D70F	Pointer --1 to WAIT
49190-49191	C026-C027	65492	FFD4	Pointer --1 to LOAD
49192-49193	C028-C029	65495	FFD7	Pointer --1 to SAVE
49194-49195	C02A-C02B	65498	FFDA	Pointer --1 to VERIFY
49196-49197	C02C-C02D	53900	D28C	Pointer --1 to DEF
49198-49199	C02E-C02F	55046	D706	Pointer --1 to POKE
49200-49201	C030-C031	51594	C98A	Pointer --1 to PRINT#

*These memory locations contain the address of the byte preceding the specified BASIC routines

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				Pointers to BASIC Routines (Continued)
49202-49203	C032-C033	51626	C9AA	Pointer -1 to PRINT
49204-49205	C034-C035	51050	C76A	Pointer -1 to CONT
49206-49207	C036-C037	50612	C5B4	Pointer -1 to LIST
49208-49209	C038-C039	50550	C576	Pointer -1 to CLR
49210-49211	C03A-C03B	51600	C990	Pointer -1 to CMD
49212-49213	C03C-C03D	65501	FFDD	Pointer -1 to SYS
49214-49215	C03E-C03F	65471	FFBF	Pointer -1 to OPEN
49216-49217	C040-C041	65474	FFC2	Pointer -1 to CLOSE
49218-49219	C042-C043	51836	CA7C	Pointer -1 to GET
49220-49221	C044-C045	50522	C55A	Pointer -1 to NEW
49222-49223	C046-C047	56133	DB45	Pointer to SGN **
49224-49225	C048-C049	56280	DBD8	Pointer to INT
49226-49227	C04A-C04B	56164	DB64	Pointer to ABS
49228-49229	C04C-C04D	0	0000	Pointer to USR pointer
49230-49231	C04E-C04F	53849	D259	Pointer to FRE
49232-49233	C050-C051	53882	D27A	Pointer to POS
49234-49235	C052-C053	56926	DE5E	Pointer to SQR
49236-49237	C054-C055	57215	DF7F	Pointer to RND
49238-49239	C056-C057	55542	D8F6	Pointer to LOG
49240-49241	C058-C059	57050	DEDA	Pointer to EXP
49242-49243	C05A-C05B	57304	DFD8	Pointer to COS
49244-49245	C05C-C05D	57311	DFDF	Pointer to SIN
49246-49247	C05E-C05F	57384	E028	Pointer to TAN
49248-49249	C060-C061	57484	E08C	Pointer to ATN
49250-49251	C062-C063	55016	D6E8	Pointer to PEEK
49252-49253	C064-C065	54870	D656	Pointer to LEN
49254-49255	C066-C067	54079	D33F	Pointer to STR$
49256-49257	C068-C069	54919	D687	Pointer to VAL
49258-49259	C06A-C06B	54885	D664	Pointer to ASC
49260-49261	C06C-C06D	54726	D5C6	Pointer to CHR$
49262-49263	C06E-C06F	54746	D5DA	Pointer to LEFT$
49264-49265	C070-C071	54790	D606	Pointer to RIGHT$
49266-49267	C072-C073	54801	D611	Pointer to MID$
49268-49297	C074-C091			Hierarchy and action addresses for operators
49298-49553	C092-C191			Table of BASIC keywords
49554-49833	C192-C2A9			BASIC error messages
				BASIC Routines
				Starting Address — Function
49834-59343	C2AA-DFFF			49834 C2AA — FOR... NEXT stack check
				49880 C2D8 — Insert line space marker
				49947 C31B — Stack overflow check
				49960 C328 — Error message abort
				50057 C389 — READY
				50091 C3AB — Handle new line

** These memory locations contain the address of the first byte of the specified BASIC routines.

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				BASIC Routines (Continued)

Starting Address		Function
50242	C442	Rechain lines after insert/delete
50287	C46F	Input line
50325	C495	Keyword encoder
50476	C52C	Line number search
50523	C55B	NEW
50551	C577	CLR
50599	C5A7	Set pointer to start of program
50613	C5B5	LIST
50776	C658	FOR
50944	C700	Statement execute
50992	C730	RESTORE
51007	C73F	STOP
51009	C741	END
51051	C76B	CONT
51077	C785	RUN
51088	C790	GOSUB
51117	C7AD	GOTO
51162	C7DA	RETURN
51200	C800	DATA
51214	C80E	Scan for next BASIC statement
51217	C811	Scan for next BASIC line
51248	C830	IF
51267	C843	REM
51283	C853	ON
51315	C873	Number fetch
51373	C8AD	LET =
51496	C928	Add ASCII digit to Accumulator #1
51595	C98B	PRINT#
51601	C991	CMD
51627	C9AB	PRINT
51740	CA1C	Print string
51769	CA39	Print character
51791	CA4F	Input data error
51837	CA7D	GET
51879	CAA7	INPUT#
51962	CAFA	Input prompt
51975	CB07	READ
52220	CBFC	Error messages
52256	CC20	NEXT
52345	CC79	Format checker
52383	CC9F	Expression evaluator
53091	CF63	DIM
53101	CF6D	Variable table lookup
53249	D001	Create new variable
53420	D0AC	Array table search/ create array

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	

BASIC Routines (Continued)

Starting Address		Function
53849	D259	FRE
53869	D26D	Integer-to-floating
53882	D27A	POS
53888	D280	Valid direct check
53901	D28D	DEF
54079	D33F	STR$
54726	D5C6	CHR$
54746	D5DA	LEFT$
54790	D606	RIGHT$
54801	D611	MID$
54870	D656	LEN
54885	D665	ASC
54919	D687	VAL
54994	D6D2	Floating-to-integer
55016	D6E8	PEEK
55047	D707	POKE
55056	D710	WAIT
55091	D733	Subtraction
55150	D76E	Addition
55542	D8F6	LOG
55607	D937	Multiplication
55704	D998	Load number to AFAC
55818	DA0A	Division
55982	DAAE	Load Accumulator (FAC)
56030	DADE	Store FAC
56072	DB08	Copy AFAC to FAC
56088	DB18	Copy FAC to AFAC
56133	DB45	SGN
56164	DB64	ABS
56280	DBD8	INT
56526	DCCE	IN line message
56553	DCE9	Numeric-to-ASCII
56319	DBFF	String-to-floating
56926	DE5E	SQR
56936	DE68	Power function
57050	DEDA	EXP
57215	DF7F	RND
57304	DFD8	COS
57311	DFDF	SIN

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				Screen Editor **Starting Address Function**
57344-5391	E000-E7FF			57384 E028 TAN 57484 E08C ATN 57593 E0F9 Subroutine to be moved to page 0 ($70-$87) 57617 E111 Initial RND seed (5 bytes) 57622 E116 Initialize BASIC system 57897 E229 Clear screen 57943 E257 Home cursor 57989 E285 Character fetch
58100-58906	E2F4-E61A			Video driver 58100 E2F4 Input from screen 58175 E33F Quote mode ($CD) switcher 58188 E34C Print character 58687 E53F Scroll 1 line
58907-59113	E61B-E6E9			Interrupt Handler
59114-59127	E6EA-E6F7			Keyboard Scan
59128-59241	E6F8-E769			Keyboard Encoding Table
59242-59391	E76A-E7FF			Subroutines for Machine Language Monitor
	Page 233-240 I/O Ports and Expansion I/O (PIA's and VIA) (59392-61439)			
				Keyboard PIA (59408-59411)
59408	E810	249	F9	I/O Port A and Data Direction register
59409	E811	60	3C	Control Register A — screen blanking 52=Screen off (blanked) 60=Screen on
59410	E812	255	FF	I/O Port B and Data Direction register 255=all keys except: 254=RVS key 253=[key 251=SPACE key 247=< key
59411	E813	61	3D	Control Registers B — #1 cassette motor 53=motor on 61=motor off
				IEEE Port PIA (59424-59427)
59424	E820	255	FF	I/O Port A and Data Direction register PEEK (59424) reads input data
59425	E821	188	BC	Control Register A — set output line CA2 POKE 59425,52=low POKE 59425,60=high
59426	E822	255	FF	I/O Port B and Data Direction registers POKE 59426, data writes output data POKE 59426,255 before a read to Port A
59427	E823	60	3C	Control Register B — set output line CB2 POKE 59427,52=low POKE 59427,60=high

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				Parallel User Port VIA (59456-59471)
59456	E840	223	DF	I/O Port B 207=#2 cassette motor on 223=#2 cassette motor off WAIT 59456,23,23 waits for vertical retrace of display Bit 1=PB1 (NFRD on IEEE connector) output line Bit 3=PB3 (ATN on IEEE connector) output line
59457	E841	255	FF	I/O Port A with handshaking
59458	E842	30	1E	Data Direction register for I/O Port B
59459	E843	0	00	Data Direction register for I/O Port A For each bit 1=output, 0=input =0 all input =255 all output
59460-59461	E844-E845	29241	7239	(Low, high order) Read Timer 1, Counter; write to Timer 1 Latch and (high byte) initiate count
59462-59463	E846-E847	65535	FFFF	(Low, high order) Read Timer 1 Latch
59464	E848	147	93	Read Timer 2 Counter low byte and reset interrupt; write to Timer 2 low byte PEEK (59464) Clock decrements every microsecond POKE 59454,n sets SR rate of shift from high (n=0) to low (n=255) for music from User Port
59465	E849	217	D9	Read Timer 2 Counter high byte; write to Timer 2 high byte and reset interrupt PEEK (59465) Clock decrements every millisecond
59466	E84A	0	00	Serial I/O Shift register (SR) POKE 59466, 15 or 85 to generate Square wave output at CB2 for playing music from User Port.
59467	E84B	0	00	Auxiliary Control register =16 Sets SR to free-running mode for music from User Port =0 for proper operation of tape drive
59468	E84C	14	0E	Peripheral Control register =12 for graphics on shifted characters =14 for lower-case letters on shifted characters
59469	E84D	0	00	Interrupt Flag register
59470	E84E	128	80	Interrupt enable register
59471	E84F	255	FF	I/O Port A without handshaking
		Page 241-256 Operating System (61440-65535)		
61440-61621	F000-F0B5			Monitor messages

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				GPIB Handler (IEEE 488 Bus)
				Starting Address Function
61622-61904	F0B6-F1D0			61622 F0B6 Setup for Listen, Talk, etc 61678 F0EE Send character 61736 F128 Output character immediate mode 61750 F136 Error messages 61796 F164 Send immediate Listen command, then secondary address 61807 F16F Output characters 61823 F17F Send Unlisten/ Untalk 61836 F18C Input character
				File Control
61905-63493	F1D1-F805			61905 F1D1 Get a character (without cursor) 61921 F1E1 Input a character (with cursor) 62002 F232 Output a character to any device 62062 F26E Close all files 62066 F272 Restore default I/O devices 62121 F2A9 CLOSE 62209 F301 STOP search 62223 F30F STOP key 62229 F315 Direct mode test 62402 F3C2 LOAD 62474 F40A Display filename/ fetch file number 62526 F43E Fetch LOAD/SAVE parameters 62560 F460 Fetch byte paramter 62566 F466 Send program name to GPIB 62612 F494 Tape header search 62647 F4B7 VERIFY 62670 F4CE Fetch OPEN/CLOSE parameters 62753 F521 OPEN 62886 F5A6 Find any tape header 62938 F5DA Write tape header 63036 F63C Process tape header 63108 F684 SYS 63134 F69E SAVE 63273 F729 Clock update 63344 F770 Set input device 63420 F7BC Set output device

Tabelle D-2. Speicherauszug für CBM BASIC 3.0, Version 3 ROMs (Forts.)

Memory Address		Sample Value		Description
Decimal	Hexadecimal	Decimal	Hexadecimal	
				Tape Control
63494-64720	F806-FCD0			63494 F806 Advance tape buffer pointer 63541 F835 Check for cassette on 63573 F855 Tape read to buffer 63622 F886 Write block to tape 63716 F8E6 Interrupt wait
				Power-on Diagnostics
64721-64784	FCD1-FD10			64721 FCD1 System reset SYS(64721) simulates power-on reset 64766 FCFE NMI interrupt entry point
64785-65471	FD11-FFBF			64769 FD01 Table of interrupt vectors
				Machine Language Monitor
				Jump Vectors
65472-65474	FFC0-FFC2	76 62753	4C F521	JMP OPEN
65475-65477	FFC3-FFC5	76 62121	4C F2A9	JMP CLOSE
65478-65480	FFC6-FFC8	76 63344	4C F770	JMP Set Input Device
65481-65483	FFC9-FFCB	76 63420	4C F7BC	JMP Set Output Device
65484-65486	FFCC-FFCE	76 62066	4C F272	JMP Restore Default I/O Devices
65487-65489	FFCF-FFD1	76 61921	4C F1E1	JMP Input Character – RDT
65490-65492	FFD2-FFD4	76 62002	4C F232	JMP Output Character – WRT
65493-65495	FFD5-FFD7	76 62402	4C F3C2	JMP LOAD
65496-65498	FFD8-FFDA	76 63134	4C F69E	JMP SAVE
65499-65501	FFDB-FFDD	76 62647	4C F4B7	JMP VERIFY
65502-65504	FFDE-FFED	76 63108	4C F684	JMP SYS
65505-65507	FFE1-FFE3	76 62223	4C F30F	JMP Test STOP Key
65508-65510	FFE4-FFE6	76 61905	4C F1D1	JMP Get Character
65511-65513	FFE7-FFE9	76 62062	4C F26E	JMP Close all files
65514-65516	FFEA-FFEC	76 63273	4C F729	JMP Clock Update
				6502 Interrupt Vectors
65530-65531	FFFA-FFFB	65766	FCFE	Non-maskable interrupt (NMI)
65532-65533	FFFC-FFFD	64721	FCD1	System reset (RESET)
65534-65535	FFFE-FFFF	58907	E61B	Interrupt request break (IRQ+BRK)

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 *)

BASIC 3.0	BASIC 4.0	Labels	Description
0000	0000	USRPOK	$4C CONSTANT AND ADDRESS TO DISPATCH USR
0000	0000	ERRNF	ERROR CALL VALUE - ECV - NEXT WITHOUT FOR
0001	0001	ADDPRC	X
0002	0002	BUFPAG	INPUT BUFFER AT $0200
0002	0002	ADDPR2	X
0003	0003	STRSIZ	NUMBER OF LOCS PER STRING DESCRIPTOR
0003	0003	INTEGR	ONE-BYTE INTEGER FROM "%INT"
0003	0003	CHARAC	STARTING DELIMITER
0004	0004	ENDCHR	ENDING DELIMITER
0004	0004	ADDPR4	X
0005	0005	COUNT	GENERAL COUNTER FOR BASIC
0006	0006	DIMFLG	FLAG TO REMEMBER DIMENSIONED VARIABLES
0007	0007	VALTYP	FLAG FOR VARIABLE TYPE 0-NUMERIC $FF-STRING
0008	0008	INTFLG	FLAG FOR INTEGER TYPE
0008	0008	ADDPR8	X
0009	0009	GARBFL	X
0009	0009	DORES	FLAG WHETHER CAN OR CAN'T CRUNCH RESERVED WORDS
000A	000A	CLMWID	SIZE OF PRINT WINDOW
000A	000A	SUBFLG	FLAG WHICH ALLOWS SUBSCRIPTS IN SYNTAX
000B	000B	INPFLG	FLAGS INPUT OR READ
000C	000C	DOMASK	MASK USED BY RELATION OPERATIONS
000C	000C	TANSGN	FLAG SIGN OF TANGENT
000D	000D	DSDESC	DS$ LENGTH AND POINTER TO DS$
000E	C010	CHANNL	ACTIVE I/O CHANNEL #
0010	0010	ERRSN	ERROR CALL VALUE - ECV - SYNTAX
0011	0011	POKER	HOLDS ADDRESS FORE POKE COMMAND
0011	0011	LINNUM	LINE NUMBER STORAGE
0012	0012	FORSIZ	AMOUNT OF BYTES USED ON STACK FOR-NEXT
0013	0013	TEMPPT	INDEX TO NEXT AVAILABLE DESCRIPTOR
0014	0014	LASTPT	POINTER TO LAST STRING TEMP LO;HI
0016	0016	TEMPST	STORAGE FOR NUMTMP TEMP DESCRIPTORS
0016	0016	ERRRG	ECV - RETURN WITHOUT GOSUB
0017	0017	NUMLEV	NUMBER OF GOSUB LEVELS ALLOWED
001E	001E	NCMPOS	X
001F	001F	INDEX	INDRIECT INDEX #1
001F	001F	INDEX1	SAME
0021	0021	INDEX2	INDIRECT INDEX #2
0023	0023	RESHO	RES -REGISTER
0024	0024	RESMOH	[
0025	0025	ADDEND	TEMP USED BY "UMULT"
0025	0025	RESMO	[
0026	0026	RESLO	[
0028	0028	LINLEN	LENGTH OF SCREEN LINE 40-COL EDITORS
0028	0028	TXTTAB	POINTER TO START OF BASIC TEXT AREA
002A	002A	VARTAB	POINTER TO START OF VARIABLES
002A	002A	ERROD	ECV - OUT OF DATA
002C	002C	ARYTAB	POINTER TO START OF ARRAY TABLE
002E	002E	STREND	POINTER TO END OF VARIABLES
0030	0030	FRETOP	POINTER TO START OF REAL STRINGS
0032	0032	FRESPC	POINTER TO TOP OF FREE STRING SPACE
0034	0034	MEMSIZ	HIGHEST RAM ADDR AVALIBLE FOR BASIC
0035	0035	ERRFC	ECV - ILLEGAL QUANTITY
0036	0036	CURLIN	CURRENT LINE BEING EXECUTED
0038	0038	OLDLIN	LAST LINE EXECUTED.(FOR CONT COMMAND)
003A	003A	OLDTXT	OLD TXTPTR (FOR CONT COMMAND) AND TEMP STORAGE
003C	003C	DATLIN	DATA LINE # FOR ERRORS

*) Für Speicherauszüge mit deutscher Kommentierung siehe Anhang E,Literatur

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
003E	003E	DATPTR	DATA STATEMENT POINTER
0040	0040	INPPTR	SOURCE OF INPUT ADDRESS
0042	0042	VARNAM	CURRENT VARIABLE NAME
0044	0044	FDECPT	POINTER INTO POWERS OF TEN FOR FOUT
0044	0044	VARPNT	POINTER TO VARIABLE IN MEMORY
0045	0045	ERROV	ECV - OVERFLOW
0046	0046	LSTPNT	PNTR TO LIST STRING
0046	0046	ANDMSK	THEN MASK USED BY WAIT FOR ANDING
0046	0046	FORPNT	POINTER TO CURRENT FOR-NEXT VARIABLE REFERENCE
0047	0047	EORMSK	THE MASK FOR EORING IN WAIT
0048	0048	VARTXT	POINTER INTO LIST OF VARIABLES
0048	0048	OPPTR	POINTER TO CURRENT OPERATOR IN TABLE
004A	004A	OPMASK	MASK CREATED BY CURRENT OPERATOR
004B	004B	GRBPNT	POINTER USED IN GARBAGE COLLECTION
004B	004B	TEMPF3	A THIRD FAC TEMPORARY 4-BYTES
004B	004B	DEFPNT	POINTER USED IN FUNCTION DEFINITION
004D	004D	DSCPNT	POINTER TO A STRING DESCRIPTION
004D	004D	ERROM	ECV - OUT OF MEMORY
0050	0050	FOUR6	VARIABLE CONSTANT USED BY GARB COLLECT
0051	0051	BUFLEN	INPUT BUFFER MAX SIZE+1
0051	0051	JMPER	$4C CONSTANT AND ADDRESS USED TO DISPATCH FUNCS
0052	0052	SIZE	X
0053	0053	OLDOV	THE OLD OVERFLOW
0054	0054	TEMPF1	A FAC TEMP 4-BYTES
0055	0055	ARYPNT	A POINTER USED IN ARRAY BUILDING
0055	0055	HIGHDS	DESTINATION OF HIGHEST ELMENT IN BLT.
0057	0057	HIGHTR	SOURCE OF HIGHEST ELEMENT TO MOVE
0059	0059	TEMPF2	A FAC TEMP 4-BYTES
005A	005A	DECCNT	NUMBER OF PLACES BEFORE DECIMAL POINT
005A	005A	LOWDS	LOCATION OF LAST BYTE TRANSFERRED INTO
005A	005A	ERRUS	ECV - UNDEF'D STATEMENT
005B	005B	TENEXP	BASE TEN EXPONENT FOR FIN AND FOUT
005C	005C	GRBTOP	A POINTER USED IN GARBAGE COLLECTION
005C	005C	DPTFLG	FLAG IF A DECIMAL POINT HAS BEEN INPUT
005C	005C	LOWTR	LAST THING TO MOVE IN BLT.
005D	005D	EXPSGN	SIGN OF BASE TEN EXPONENT
005D	005D	EPSGN	X
005E	005E	DSCTMP	THIS IS WHERE TEMP DESCS ARE BUILT
005E	005E	FAC	THE MAIN FLOATING POINT ACCUMULATOR
005E	005E	FACEXP	THE EXPONENT BYTE
005F	005F	FACHO	[MOST SIGNIFICANT BYTE OF MANTISSA
0060	0060	FACMOH	[ONE MORE
0061	0061	INDICE	INDICE IS SET UP HERE BY "GINT"
0061	0061	FACMO	[MIDDLE ORDER OF MANTISSA
0062	0062	FACLO	[LEAST SIG BYTE OF MANTISSA
0063	0063	FACSGN	SIGN OF FAC (0 OR -1) WHEN UNPACKED
0064	0064	DEGREE	A CONT USED BY POLYNOMIALS
0064	0064	SGNFLG	SIGN OF FAC IS PRESERVED HERE BY FIN
0065	0065	BITS	COUNTER FOR # OF BIT SHIFTS TO NORMALIZE FAC
0066	0066	ARGEXP	THE ARG REGISTER EXPONENT
0067	0067	ARGHO	[
0068	0068	ARGMOH	[
0069	0069	ARGMO	[
006A	006A	ARGLO	[
006B	006B	ARGSGN	THE SIGN (SAME AS FAC)
006B	006B	ERRBS	ECV - BAD SUBSCRIPT

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
006C	006C	STRNG1	POINTER TO A STRING OR DESCRIPTOR
006C	006C	ARISGN	A SIGN REFLECTING THE RESULT
006D	006D	FACOV	OVERFLOW BYTE OF THE FAC
006E	006E	BUFPTR	POINTER TO BUF USED BY "CRUNCH ROUTINE"
006E	006E	STRNG2	POINTER TO STRING OR DESC.
006E	006E	POLYPT	POINTER INTO POLYNOMIAL COEFFICIENTS.
006E	006E	CURTOL	ABSOLUTE LINEAR INDEX IS FORMED HERE
006E	006E	FBUFPT	POINTER INTO FBUFFER USED IN FOUT.
0070	0070	CHRGET	ROUTINE - GETS NEXT CHARACTER FROM BASIC TEXT
0076	0076	CHRGOT	ROUTINE -REGETS CURRENT CHARACTER FROM BASIC TEXT
0077	0077	TXTPTR	POINTER TO CURRENT SOURCE TEXT
0078	0078	ERROD	ECV - REDIM'D ARRAY
007D	007D	0NUM	LABEL IN CHRGET
0080	0080	ENDTK	TOKEN - END
0081	0081	FORTK	TOKEN - FOR
0083	0083	DATATK	TOKEN - DATA
0085	0085	ERRDVO	ECV - DIVISION BY ZERO
0087	0087	CHRRTS	LABEL IN CHRGET
0088	0088	RNDX	NEXT RANDOM NUMBER - INITIAL LOAD FROM ROM
0089	0089	GOTOTK	TOKEN - GOTO
008B	008B	ZZ7	X
008D	008D	CTIMR	24 HR CLOCK 1/60 OF SEC
008D	008D	GOSUTK	TOKEN - GOSUB
008F	008F	REMTK	TOKEN - REM
0095	0095	ERRID	ECV - ILLEGAL DIRECT
0096	0096	CSTAT	I/O OPERATION STATUS BYTE (VARIABLE ST)
0099	0099	PRINTK	TOKEN - PRINT
00A2	00A2	SCRATK	TOKEN - NEW
00A3	00A3	TABTK	TOKEN - TAB
00A3	00A3	ERRTM	ECV - TYPE MISMATCH
00A4	00A4	TOTK	TOKEN - TO
00A5	00A5	FNTK	TOKEN - FN
00A6	00A6	SPCTK	TOKEN - SPC
00A7	00A7	THENTK	TOKEN - THEN
00A8	00A8	NOTTK	TOKEN - NOT
00A9	00A9	STEPTK	TOKEN - STEP
00AA	00AA	PLUSTK	TOKEN - +
00AB	00AB	MINUTK	TOKEN - -
00B0	00B0	ERRLS	ECV - STRING TO LONG
00B1	00B1	GREATK	TOKEN - >
00B2	00B2	EQULTK	TOKEN - =
00B3	00B3	LESSTK	TOKEN - <
00B4	00B4	ONEFUN	TOKEN - SGN START OF SINGLE PARM FUNCTIONS
00BF	00BF	ERRBD	ECV - FILE DATA
00C6	00C6	TRMPOS	X
00C7	00C7	LASNUM	TOKEN - CHR$ LAST FUNC WITH ARITHMETIC PARMS
00C8	00C8	ERRST	ECV - FORMULA TOO COMPLEX
00CB	00CB	GOTK	TOKEN - GO (GO TO)
00DB	00DB	ERRCN	ECV - CAN'T CONTINUE
00E9	00E9	ERRUF	ECV - UNDEF'D FUNCTION
00FF	00FF	PI	VALUE OF PI SYMBOL
00FF	00FF	LOFBUF	START OF FOUT STRING FOR STRD AND TI$
0100	0100	FBUFFR	FOUT BUFFER HOLDS ASCII STRING FOR OUTPUT
01FB	01FB	STKEND	TOP OF STACK FOR BASIC
01FF	01FF	ZZ1	X
01FF	01FF	ZZ5	X

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
01FF	01FF	ZZ4	X
0200	0200	BUF	BASIC INPUT BUFFER (80 CHARACTERS-BYTES LONG)
0200	0200	BUFOFS	SAME AS ABOVE
0201	0201	ZZ2	X
0202	0202	ZZ3	X
0400	0400	RAMLOC	BEGINING OF RAM AVALIABLE FOR BASIC TEXT
0000	8000	OFFSET	*VALUE USED IN ASSEMBLY - ROM VERSION
0000	8B88	ZZ8	X
C000	B000	ROMLOC	BEGINING OF BASIC ROMS -V2=$C000 V4=$B000
C000	B000	STMDSP	START OF COMMAND DISPATCH TABLE
C046	B066	FUNDSP	START OF FUNCTION DISPATCH TABLE
C04C	B06C	USRLOC	X
C074	B094	OPTAB	START OF MATH OPERATORS DISPATCH TABLE
C089	B0A9	NEGTAB	UNITARY NEGATE DISPATCH (.BYTE 125,DISPATCH)
C08C	B0AC	NOTTAB	NOT OPERATOR DISPATCH (.BYTE 90,DISPATCH)
C08F	B0AF	PTDORL	COMPARISON DISPATCH (.BYTE 100,DISPATCH)
C092	B0B2	RESLST	START OF RESERVED WORD LIST (ASCII,END(OR $80))
C192	B20D	ERRTAB	START OF BASIC ERROR MESSAGE STORAGE
C28B	B306	ERR	MESSAGE - "ERROR"
C292	B30D	INTXT	MESSAGE - "IN"
C297	B312	REDDY	MESSAGE - "READY"
C2A2	B31B	BRKTXT	MESSAGE - "BREAK"
C2AA	B322	FNDFOR	PEEKS AT THE STACK FOR AN ACRTIVE "FOR" LOOP
C2AF	B327	FFLOOP	X
C2C4	B33C	CMPFOR	X
C2D0	B348	ADDFRS	X
C2D7	B34F	FFRTS	X
C2D8	B350	BLTU	"OPENS UP" A SPACE IN BASIC FOR A NEW LINE
C2DF	B357	BLTUC	X
C2FC	B374	BLT1	X
C308	B380	BLTLP	X
C30C	B384	MOREN1	X
C313	B38B	DECBLT	X
C31B	B393	GETSTK	TEST FOR STACK-TOO-DEEP ERROR
C328	B3A0	REASON	CHECKS FOR AVALIABLE MEMORY SPACE
C332	B3AA	TRYMOR	X
C336	B3AE	REASAV	X
C341	B3B9	REASTO	X
C354	B3CC	REARTS	X
C355	B3CD	OMERR	OUT OF MEMORY ERROR VECTOR
C357	B3CF	ERROR	ERROR HANDLER (ERROR TYPE IN .X)
C364	B3DA	ERRCRD	X
C36A	B3E0	GETERR	X
0000	B3ED	TYPERR	PRINTS OUT THE ERROR MESSAGE
C37E	B3F4	ERRFIN	X
C389	B3FF	READY	PRINTS "READY." GOES INTO MAIN BASIC LOOP (← NMI)
C392	B406	MAIN	MAIN BASIC LOOP, ANALYZES INPUT LINES
C3AB	B41F	MAIN1	LINES THAT START WITH A NUMBER HANDLED HERE
C3E6	B45A	@DECT1	X
C3EE	B462	MLOOP	X
C3FC	B470	NODEL	X
C417	B48B	NODELC	X
C431	B4A5	STOLOP	X
C439	B4AD	FINI	CLEANS BASIC SYSTEM UP; CLR
C442	B4B6	LNKPRG	RELINKS BASIC STATEMENTS IN TEXT AREA
C44B	B4BF	CHEAD	X

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
C453	B4C7	CZLOOP	X
C46E	B4E1	LNKRTS	X
C46F	B4E2	INLIN	INPUT A LINE OF INFORMATION INTO BUF (MAX 80 CHARS)
C471	B4E4	INLINC	X
C47E	B4F8	FININ1	X
C495	B4FB	CRUNCH	LOOKS UP KEYWORDS IN AN INPUT LINE
C49B	B501	KLOOP	X
C4A7	B50D	CMPSPC	X
C4BD	B523	KLOOP1	X
C4C5	B52B	MUSTCR	X
C4CF	B53D	RESER	X
C4D1	B544	RESCON	X
C4E0	B552	GETBPT	X
C4E2	B554	STUFFH	X
C4F5	B567	COLIS	X
C4F7	B569	NODATT	X
C4FE	B570	STR1	X
C507	B579	STRNG	X
C50E	B580	NTHIS	X
C512	B584	NTHIS1	X
0000	B58D	NTHIS2	X
C522	B599	CRDONE	X
C52C	B5A3	FNDLIN	SEARCHES FOR A LINE NUMBER (NUMBER IN LINNUM)
C530	B5A7	FNDLNC	X
C547	B5BE	FNDLO1	X
C550	B5C7	AFFRTS	X
C559	B5D0	FLINRT	X
C55A	B5D1	FLNRTS	X
C55B	B5D2	SCRATH	IMPLEMENTS "NEW" COMMAND - CLEARS EVERY THING
C55D	B5D4	SCRTCH	X
C572	B5E9	RUNC	X
C577	B5EE	CLEAR	CLR - ROUTINE
C579	B5F0	CLEARC	X
0000	B60B	FLOAD	X
C593	B60E	STKINI	X
C5A6	B621	STKRTS	X
C5A7	B622	STXTPT	TXTPTR=TXTTAB-1
C5B5	B630	LIST	ROUTINE - LIST
C5BD	B638	GOLST	X
C5D4	B64F	LSTEND	X
C5E2	B65D	LIST4	X
C5FF	B67A	TSTDUN	X
C601	B67C	TYPLIN	X
C608	B683	PRIT4	X
C60C	B687	PLOOP	X
C619	B694	PLOOP1	X
C62D	B6A8	GRODY	X
C630	B6AB	QPLOP	X
C642	B6C5	RESRCH	X
C645	B6C8	RESCR1	X
0000	B6CE	RESCR2	X
C64D	B6D4	PRIT3	X
0000	B6D5	PRIT3B	X
C658	B6DE	FOR	ROUTINE - FOR
C669	B6EF	NOTOL	X
C6A1	B727	LDFONE	X

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
C6B5	B73B	ONEON	X
C6C4	B74A	NEWSTT	MAIN STATEMENT DISPATCH LOOP (DO NEXT STATEMENT)
C6D4	B759	DIRCON	X
C6E4	B769	DIRCN1	X
C6F7	B77C	GONE	DISPATCHES NEXT BYTE CHRGET RETURNS
C700	B785	GONE3	DISPATCHES .A IF NONZERO ELSE LOOP TO NEWSTT
C702	B787	GONE2	X
0000	B795	GONE4	X
C717	B7A2	GLET	X
C71A	B7A5	MORSTS	X
C71E	B7A9	SNERR1	SYNTAX ERROR VECTOR
0000	B7AC	GO	HANDLE GO TOKEN CASE (FIND A TO)
C730	B7B7	RESTOR	ROUTINE - RESTORE
C73A	B7C1	RESFIN	X
C73E	B7C5	ISCRTS	X
C73F	B7C6	STOP	STOP - SEC END - CLC
C741	B7C8	END	ROUTINE - END
C742	B7C9	STOPC	ROUTINE - STOP
C751	B7D8	STPEND	X
C759	B7E0	DIRIS	X
C75B	B7E2	ENDCON	X
C768	B7EB	GORDY	JMP READY
C76B	B7EE	CONT	ROUTINE - CONT
C784	B807	CONTRT	X
C785	B808	RUN	ROUTINE - RUN
C790	B813	GOSUB	ROUTINE - GOSUB
C7A4	B827	RUNC2	X
C7AD	B830	GOTO	ROUTINE - GOTO
C7C4	B847	LUK4IT	X
C7C8	B84B	LUKALL	X
C7D9	B85C	GORTS	X
C7DA	B85D	RETURN	ROUTINE - RETURN
C7EB	B86E	USERR	BAD SUBSCRIPT ERROR VECTOR
C7F0	B873	SNERR2	SYNTAX ERROR VECTORY
C7F3	B876	RETU1	X
C800	B883	DATA	X
C803	B886	ADDON	X
C80D	B890	REMRTS	X
C80E	B891	DATAN	SEARCH FOR NEXT '
C811	B894	REMN	LOOK FOR EOL($00) (TXTPTR OFFSET IN .Y)
C819	B89C	EXCHQT	X
C821	B8A4	REMER	X
C830	B8B3	IF	ROUTINE - IF
C83F	B8C2	OKGOTO	X
C843	B8C6	REM	ROUTINE - REM
C848	B8CB	DOCOND	X
C850	B8D3	DOCO	X
C853	B8D6	ONGOTO	ROUTINE - ON (GOTO OR GOSUB)
C85B	B8DE	SNERR3	SYNTAX ERROR VECTOR
C85F	B8E2	ONGLOP	X
C867	B8EA	ONGLP1	X
C872	B8F5	ONGRTS	X
C873	B8F6	LINGET	INPUT A BASIC LINE NUMBER (0-63999)(VALUE IN LINNUM)
C879	B8FC	MORLIN	X
C8A7	B92A	NXTLGC	X
C8AD	B930	LET	ROUTINE - LET

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
C8CA	B94D	GINTGR	X
C8DE	B961	COPFLT	X
C8E1	B964	COPSTR	X
C8E2	B965	INPCOM	X
C8F5	B978	TIMELP	X
C90F	B992	NOML6	X
C91F	B9A2	TIMEST	X
C928	B9AB	TIMNUM	X
C92F	B9B2	FCERR2	ILLEGAL QUANITY ERROR VECTOR
C932	B9B5	GOTNUM	X
C937	B9BA	GETSPT	COPY STRINGS IF NEEDED
0000	B9BE	DSKX0	X
0000	B9D2	DSKX1	X
0000	B9D4	DSKX2	X
C948	B9E1	GVARIA	X
C956	B9EF	DNTCPY	X
C95D	B9F6	COPY	X
C973	BA13	COPYC	X
0000	BA2E	COPY00	X
0000	BA44	COPY01	X
0000	BA46	COPY02	X
0000	BA4E	STRADJ	POINT TO STRING FOR A COPY
0000	BA6C	ADJ	X
0000	BA70	ADJXX	X
0000	BA74	ADJ02	X
0000	BA83	ADJ00	X
0000	BA85	ADJ01	X
C98B	BA88	PRINTN	ROUTINE - PRINT#
C991	BA8E	CMD	ROUTINE - CMD
C99B	BA98	SAVEIT	X
C9A5	BAA2	STRDON	X
C9A8	BAA5	NEWCHR	X
C9AB	BAA8	PRINT	ROUTINE - PRINT
C9AD	BAAA	PRINTC	X
C9D5	BAD2	FININL	X
C9E2	BADF	CRDO	OUTPUT A CARRIAGE RETURN
C9EC	BAED	CRFIN	X
C9EE	BAEF	PRTRTS	X
C9EF	BAF0	COMPRT	X
C9F2	BAF3	MORCO1	X
C9FC	BAFD	TABER	TAB AND SPC HANDLER
CA0C	BB0D	ASPAC	X
CA0D	BB0E	XSPAC	X
CA0E	BB0F	XSPAC2	X
CA11	BB12	NOTABR	X
CA17	BB18	XSPAC1	X
CA1C	BB1D	STROUT	PRINT STRING FROM ADDRESS IN .Y AND .A
CA1F	BB20	STRPRT	PRINT STRING POINTED TO BY INDEX
CA26	BB27	STRPR2	X
CA39	BB3A	OUTSPC	OUTPUT A SPACE
CA40	BB41	CRTSKP	OUTPUT A $1D
CA43	BB44	OUTQST	OUTPUT A ?
CA45	BB46	OUTDO	OUTPUT THE CHAR IN .A
CA4C	BB49	OUTRTS	X
CA4F	BB4C	TRMNOK	HANDLES BAD INPUT DATA
CA59	BB56	GETDTL	X

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
CA5D	BB5A	STCURL	X
CA61	BB5E	SNERR4	SYNTAX ERROR VECTOR
CA64	BB61	TRMNO1	X
CA6D	BB6A	DOAGIN	X
CA7D	BB7A	GET	ROUTINE - GET OR GET#
CA94	BB91	GETTTY	X
CAA7	BBA4	INPUTN	ROUTINE - INPUT#
CAB7	BBB4	IODONE	RESTORE INPUT TO KEYBOARD
CAB9	BBB6	IORELE	X
CAC1	BBBE	INPUT	ROUTINE - INPUT
CAD2	BBCD	NOT@TI	X
CADA	BBD5	GETAGN	X
CAED	BBE8	BUFFUL	X
0000	BBF1	PTHRTI	X
CAFA	BBF5	@INLIN	PROMPTS AND RECEIVES THE INPUT
CB04	BBFF	GINLIN	X
CB07	BC02	READ	ROUTINE - READ
CB0E	BC09	INPCON	X
CB10	BC0B	INPCO1	X
CB16	BC11	INLOOP	X
CB42	BC3D	@DATA	X
CB4B	BC46	GETNTH	X
CB4E	BC49	DATBK	X
CB52	BC4D	DATBK1	X
CB66	BC61	SET@UT	X
CB72	BC6D	RESETC	X
CB73	BC6E	NOWGET	X
CB7E	BC79	NOWGE1	X
CB8A	BC85	NUMINS	X
CB92	BC8D	STRDN2	X
CB9E	BC99	TRMOK	X
CBB9	BCB4	DATLOP	X
CBD2	BCCD	NOWLIN.	X
CBDF	BCDA	VARFND	X
CBEA	BCE5	VARY@	PRINT "EXTRA IGNORED " IF KEYBOARD AND A SEPERATOR
CBFB	BCF6	INPRTS	X
CBFC	BCF7	EXIGNT	MESSAGE - EXTRA IGNORED
CC0D	BD07	TRYAGN	MESSAGE - ?REDO FROM START
CC20	BD19	NEXT	ROUTINE - NEXT
CC26	BD1F	GETFOR	X
CC29	BD22	STXFOR	X
CC34	BD2D	ERRGOS	X
CC36	BD2F	HAVFOR	X
CC76	BD6F	NEWSGO	X
CC79	BD72	LOOPDN	CHECKS DATA FORMAT
CC8B	BD84	FRMNUM	JMP FRMEVL
CC8E	BD87	CHKNUM	CHECK THAT CURRENT TYPE IS NUMERIC
CC90	BD89	CHKSTR	CHECK THAT CURRENT TYPE IS STRING (CHKS VALTYP)
CC91	BD8A	CHKVAL	X
CC97	BD90	CHKOK	X
CC98	BD91	DOCSTR	X
CC9A	BD93	CHKERR	TYPE MISMATCH ERROR VECTOR
CC9C	BD95	ERRGO4	X
CC9F	BD98	FRMEVL	FORMULA EVALUATOR - EVALUATES ALL FORMULAS
CCA5	BD9E	FRMEV1	X
CCAA	BDA3	LPOPER	X

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
CCB9	BDB2	TSTOP	X
CCBC	BDB5	LOPREL	X
CCD6	BDD1	ENDREL	X
CCF1	BDEA	OPREG	X
CCFA	BDF3	DOPREC	X
CCFB	BDF4	NEGPRC	X
CD08	BE01	FINREL	X
CD12	BE0B	FINRE2	X
CD1A	BE13	OPREC1	X
CD21	BE1A	DOPRE1	PUSHES A PARTIAL EVALUATION ON THE STACK
CD31	BE2A	SNERR5	SYNTAX ERROR VECTOR
CD34	BE2D	PUSHF1	X
CD39	BE32	PUSHF	X
CD44	BE41	FORPSH	X
CD59	BE56	OOP	X
CD5C	BE59	OOPGO	X
CD5E	BE5B	OCHNUM	X
CD65	BE62	UNPSTK	X
CD67	BE64	PULSTK	RESTORE ARG FROM STACK (PUSHED EVALUATION)
CD81	BE7E	OOPRTS	X
CD83	BE80	UNPRTS	X
CD84	BE81	EVAL	EVALUATES NUMERIC FORMULAS
CD88	BE85	EVAL0	X
CD8D	BE8A	EVAL1	X
CD90	BE8D	EVAL2	X
CDA3	BEA0	PIVAL	STORAGE - THE BINARY.VALUE OF PI
CDA8	BEA5	ODOT	X
CDB8	BEB5	STRTXT	IMMEDIATE STRINGS HANDLER
CDC1	BEBE	STRTX2	X
CDC7	BEC4	EVAL3	X
CDCF	BECC	NOTOP	EVAL - NOT
CDDE	BEDB	EVAL4	X
CDEC	BEE9	PARCHK	EVALUATE A FUNCTION WITHIN ()'S (FRMEVL)
CDF2	BEEF	CHKCLS	CHECK FOR RIGHT PARENTHESIS)
CDF5	BEF2	CHKOPN	CHECK FOR LEFT PARENTHESIS (
CDF8	BEF5	CHKCOM	CHECK FOR A COMMA
CDFA	BEF7	SYNCHR	COMPARE TXTPTR AGAINST .A IF <> THEN...
CE03	BF00	SNERR	...SYNTAX ERROR VECTOR
CE08	BF05	DOMIN	SET UP FUNCTION FOR FUTURE EVALUATION
CE0A	BF07	GONPRC	X
0000	BF0C	CKSMB0	THE CHECKSUM BYTE FOR THE $B000 ROM
0000	BF0D	ISVJMP	JMP ISVAR
0000	BF10	PABB0	PATCHES
0000	BF10	PATCHG	P
0000	BF1D	PCTH0	P
0000	BF1E	PCTH1	P
0000	BF21	PATCHH	P
0000	BF2E	PATCHI	P
CE0F	BF8C	ISVAR	SET UP A VARIABLE NAME SEARCH
CE11	BF8E	ZZ6	X
CE12	BF8F	ISVRET	X
0000	BFC1	ISVDS	DS$ TEST AND HANDLER
CE42	BFD3	STRRTS	X
CE43	BFD4	GOOO	X
CE54	BFE5	GOOOOO	X
0000	BFFC	CHKDS	CHECK FOR A DS VARIABLE

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
CE69	C003	SETTIM	ASSIGN TIME TO TI
CE75	C00F	OSTATU	"
0000	C01C	OOSAV	"
CE82	C040	GOMOVF	"
CE89	C047	ISFUN	DISPATCH AND EVAL IF IT'S A FUNCTION
CEB3	C071	OKNORM	"
CEB8	C076	FINGO	PLACE FUNCTIONS DISPATCH ADDRESS IN JUMPER AND GO
CEC8	C086	OROP	EVAL - OR
CECB	C089	ANDOP	EVAL - AND
CEF8	C0B6	DOREL	DO COMPARISONS
CF10	C0CE	STRCMP	"
CF38	C0F6	STASGN	"
CF3D	C0FB	NXTCMP	"
CF43	C101	QCOMP	"
CF48	C106	GETCMP	"
CF54	C112	DOCMP	"
CF5D	C11B	GOFLOT	"
CF60	C11E	DIM3	MULTIPLE DIM RE-ENTRY (CHKS FOR A COMMA)
CF63	C121	DIM	ROUTINE - DIM
CF6D	C12B	PTRGET	SEARCHES FOR A BASIC VARIABLE
CF72	C130	PTRGT1	"
CF74	C132	PTRGT2	"
CF7E	C13C	INTERR	SYNTAX ERROR VECTOR
CF81	C13F	PTRGT3	"
CF91	C14F	ISSEC	"
CF92	C150	EATEM	"
CF9C	C15A	NOSEC	"
CFA6	C164	NOTSTR	"
CFB6	C174	TURNON	"
CFBD	C17B	STRNAM	"
CFD3	C18F	STXFND	"
CFD5	C191	LOPFND	"
CFDF	C19B	LOPFN	"
0000	C1AB	NXTPTR	MOVE SEARCH TO NEXT TABLE ENTRY
CFED	C1AC	NOTIT	"
CFF7	C1B6	ISLETC	"
D000	C1BF	ISLRTS	"
D001	C1C0	NOTFNS	DID NOT FIND VARIABLE - CREATE A NEW ONE
D007	C1C6	LOZR	"
D00C	C1CB	NOTEVL	"
D01C	C1DB	GOBADV	"
D01F	C1DE	OSTAVR	CHECK FOR ST CASE
0000	C1E6	ODSVAR	CHECK FOR DS CASE
D027	C1F2	VAROK	GOOD USABLE VARIABLE
D03D	C208	NOTEVE	"
D448	C21C	ARYVA2	"
D44C	C220	ARYVA3	"
D457	C228	ARYVGO	SEARCH THE ARRAYS
D488	C259	ARYGET	MOVE THRU THE ARRAY TABLES
D492	C263	GOGO	"
0000	C281	GOGO1	"
D4D0	C290	OVARTB	"
0000	C29D	ARYDON	"
D069	C2B9	FINPTR	LOGS BASIC VARIABLE LOCATION
D073	C2C3	FINNOW	"
D078	C2C8	FMAPTR	ARRAY POINTER SUBROUTINE

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
D084	C2D4	JSRGM	X
D089	C2D9	N32768	STORAGE - THEN BINARY VALUE -32768
D08D	C2DD	INTIDX	EVALUATE FORMULA RESULT IS POSITIVE INTEGER VALUE
D093	C2E3	POSINT	CONVERT FLOATING BINARY TO POSITVE INTEGER
D09A	C2EA	AYINT	CONVERT FLOATING BINARY TO INTEGER
D0A7	C2F7	NONONO	ILLEGAL QUANITY ERROR VECTOR
D0A9	C2F9	QINTGO	JMP QINT
D0AC	C2FC	ISARY	LOCATES AND/OR CREATES ARRAYS
D0B6	C306	INDLOP	X
D0F7	C347	LOPFDA	X
D103	C353	LOPFDV	X
D112	C362	NMARY1	X
D120	C370	BSERR	BAD SUBSCRIPT ERROR VECTOR
D123	C373	FCERR	ILLEGAL QUANITY ERROR VECTOR
D125	C375	ERRGO3	X
D128	C378	GOTARY	X
D130	C380	NOTFDD	X
D150	C39F	NOTFLT	X
D159	C3A8	STOMLT	X
D162	C3B1	LOPPTR	X
D172	C3C1	NOTDIM	X
D195	C3E4	GREASE	X
D1A4	C3F3	ZERITR	X
D1A9	C3F8	DECOUR	X
D1C6	C415	GETDEF	X
D1CE	C41D	INLPNM	X
D1E4	C433	BSERR2	SYNTAX ERROR VECTOR
D1E7	C436	OMERR1	OUT OF MEMORY ERROR VECTOR
D1EA	C439	INLPN2	X
D1EB	C43A	INLPN1	X
D1FC	C44B	ADDIND	X
D20D	C45C	NOTFL1	X
D213	C462	STOML1	X
D227	C476	DIMRTS	X
D228	C477	UMULT	INTEGER ARITHMETIC ROUTINES FOR MULTI-DIM ARRAYS
D231	C480	UMULTD	X
D23B	C48A	UMULTC	X
D254	C4A3	UMLCNT	X
D258	C4A7	UMLRTS	X
D259	C4A8	FRE	ROUTINE - FRE(X)
D260	C4AF	NOFREF	X
D26D	C4BC	GIVAYF	CONVERTS INTEGER TO FLOATING BINARY
D27A	C4C9	POS	ROUTINE - POS(X)
D27C	C4CB	SNGFLT	X
D280	C4CF	ERRDIR	IF COMMAND TYPE IS INDIRECT ONLY - ILLEGAL DIRECT
D288	C4D7	ERRGUF	UNDEFINED FUNCTION ERROR VECTOR
D28D	C4DC	DEF	ROUTINE - DEF FN()=
D2BB	C50A	GETFNM	X
D2CE	C51D	FNDOER	EVALUATES FN() IN FORMULAS
D2F2	C541	DEFSTF	X
D329	C578	DEFFIN	X
D33F	C58E	STRD	ROUTINE - STR$
D349	C598	TIMSTR	MAKE A STRING OUT OF INFO AT $01FF
D34F	C59E	STRINI	MAKE A STRING OUT OF (FACMO POINTER)
D357	C5A6	STRSPA	X
D361	C5B0	STRLIT	SCANS AND SETS UP STRING ELEMENTS

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
D367	C5B6	STRLT2	X
D371	C5C0	STRGET	X
D37E	C5CD	STRFIN	X
D382	C5D1	STRFI1	X
D383	C5D2	STRFI2	X
D38F	C5DE	STRST2	X
D399	C5E8	STRCP	X
D3A4	C5F3	PUTNEW	CHECK STRING TEMPS PLACE DATA IN TEMPS
D3AC	C5FB	ERRGO2	X
D3AF	C5FE	PUTNW1	X
D3CE	C61D	GETSPA	BUILDS STRING VECTORS
D3D0	C61F	TRYAG2	X
D3DB	C62D	TRYAG3	X
0000	C63A	TRYAG4	X
D3E5	C644	STRFRE	X
0000	C65A	GETRTS	X
D3F0	C65B	GARBAG	X
D400	C66A	GARBA2	DOES 'GARBAGE COLLECTION' - PACKS STRINGS
0000	C67E	GLOOP	X
0000	C68A	COL00	X
0000	C693	COL00B	X
0000	C69E	COL00A	X
0000	C6A9	COL01	X
0000	C6B2	COL02	X
0000	C6CE	GLOP1	X
0000	C6D8	COL02B	X
0000	C6F0	COL02A	X
0000	C700	GRBEND	JMP ENDGRB
0000	C703	COL03	MOVES FRESPC TO FRETOP
0000	C716	ENDGRB	MOVES FRESPC TO FRETOP
0000	C71F	SKIP2	X
0000	C724	SKIP2A	X
0000	C726	MOVPNT	X
0000	C730	MOV00	X
0000	C735	MOVTOP	X
0000	C73F	MOV01	X
0000	C744	SETINX	X
0000	C746	SET00	X
D517	C74F	CAT	CONCATENATE TWO STRINGS (FAC) AND (++(TXTPTR))
D537	C76F	SIZEOK	X
D554	C78C	MOVINS	X
D562	C79A	MOVSTR	X
D566	C79E	MOVDO	X
D56A	C7A2	MOVLP	X
D573	C7AB	MVDONE	X
D570	C7B4	MVSTRT	X
D57D	C7B5	FRESTR	X
D580	C7B8	FREFAC	X
D584	C7BC	FRETMP	FREES UP TEMPORARY STRING POINTERS
0000	C7DE	RE300	X
0000	C7F6	FRE01	X
D5AF	C7FC	FREPLA	X
0000	C7FE	FRE02	X
D5B5	C811	FRETMS	X
D5C5	C821	FRERTS	X
D506	C822	CHRD	ROUTINE - CHR$(VALUE) (VALUE 0-255)

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
D5DA	C836	LEFTD	ROUTINE - LEFT$()
D5E0	C83C	RLEFT	X
D5E6	C842	RLEFT1	X
D5E7	C843	RLEFT2	X
D5E8	C844	RLEFT3	X
D5FF	C85B	PULMOR	X
D606	C862	RIGHTD	ROUTINE - RIGHT$()
D611	C86D	MIDD	ROUTINE - MID$()
D622	C87E	MID2	X
D63B	C897	PREAM	USED BY RIGHT
D656	C8B2	LEN	ROUTINE - LEN(STRING)
D65C	C8B8	LEN1	X
D665	C8C1	ASC	ROUTINE - ASC(STRING)
D672	C8CE	GOFUC	X
D675	C8D1	GTBYTC	DOES A CHRGET AND GETBYT
D678	C8D4	GETBYT	EVALUATE THE FORMULA AND RETURN A BYTE VALUE (IN .X)
D67B	C8D7	CONINT	X
D687	C8E3	VAL	ROUTINE - VAL(STRING)
D6A7	C903	VAL2	X
D6BD	C918	ST2TXT	X
D6C5	C920	VALRTS	X
D6C6	C921	GETNUM	EVALUATE FORMULA AND RETURN INTEGER VALUE (0-65535)
D6CC	C927	COMBYT	X
D6D2	C92D	GETADR	CONVERT FAC TO VALUE(0-65535) PLACE IN POKER
D6E8	C943	PEEK	ROUTINE - PEEK(X)
D6FB	C94E	GETCON	X
D6FE	C951	DOSGFL	X
D707	C95A	POKE	ROUTINE - POKE X
D710	C963	FNWAIT	ROUTINE - WAIT
D71F	C972	STORDO	X
D723	C976	WAITER	X
D72B	C97E	ZERRTS	X
D72C	C97F	FADDH	ADD 1/2 TO FPB VALUE IN FAC
D733	C986	FSUB	UNPACKS ARGUMENT AND SUBTRACT FPB
D736	C989	FSUBT	FPB SUBRATRACTION ARG-FAC
D76E	C998	FADD5	X
D773	C99D	FADD	UNPACK ARGUMENT INTO ARG DO A FPB ADD
D776	C9A0	FADDT	FPB ADDITION FAC=FAC+ARG
D783	C9AD	FADDC	X
D79F	C9C9	FADDA	X
D7A3	C9CD	FADD1	X
D7AF	C9D9	FADD4	X
D7BB	C9E5	SUBIT	X
D7DE	CA08	FADFLT	X
D7E3	CA0D	NORMAL	NORMALIZE ADDITION AND SUBTRACTION RESULTS
D7E7	CA11	NORM3	X
D803	CA2D	ZEROFC	FAC=0
D805	CA2F	ZEROF1	X
D807	CA31	ZEROML	MAKE SIGN POSITIVE
D80A	CA34	FADD2	X
D829	CA53	NORM2	X
D835	CA5F	NORM1	X
D842	CA6C	SQUEE2	X
D844	CA6E	RNDSHF	X
D852	CA7C	RNDRTS	X
D853	CA7D	NEGFAC	CONFLEMENT FAC ENTIRELY

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
D859	CA83	NEGFCH	COMPLEMENT JUST THE NUMBER IN FAC
D87B	CAA5	INCFAC	INCREMENT FAC
D889	CAB3	INCFRT	X
D88A	CAB4	OVERR	OVERFLOW ERROR VECTOR
D88F	CAB9	MULSHF	SHIFER ROUTINES
D891	CABB	SHFTR2	X
D8A5	CACF	SHIFTR	X
D8B2	CADC	SHFTR3	X
D8B8	CAE2	SHFTR4	X
D8BC	CAE6	ROLSHF	X
D8C6	CAF0	SHFTRT	X
D8C8	CAF2	FONE	FLOATING-POINT-BINARY CONSTANTS
D8CD	CAF7	LOGCN2	X
D8E2	CB0C	S@R05	X
D8E7	CB11	S@R20	X
D8EC	CB16	NEGHLF	X
D8F1	CB1B	LOG2	X
D8F6	CB20	LOG	ROUTINE - LOG(X)
D8FD	CB27	LOGERR	ILLEGAL QUANITY ERROR VECTOR
D900	CB2A	LOG1	X
0000	CB5A	MULLN2	X
D934	CB5E	FMULT	FPB MULTIPLY FAC=FAC*ARG
D937	CB61	FMULTT	FPB MULTIPLY WITH ARG AND .AC LOADED
D965	CB8F	MLTPLY	X
D96A	CB94	MLTPL1	X
D96D	CB97	MLTPL2	X
D989	CBB3	MLTPL3	X
D997	CBC1	MULTRT	X
D998	CBC2	CONUPK	UNPACK MEMORY INTO ARG
D9C3	CBED	MULDIV	CHECK AND ADJUST EXPS OF FPB MULT AND DIV
D9C5	CBEF	MLDEXP	X
D9D0	CBFA	TRYOFF	X
D9E0	CC0A	MLOVEX	X
D9E6	CC10	ZEREMV	X
D9EB	CC15	GOOVER	OVERFLOW ERROR VECTOR
D9EE	CC18	MUL10	MULTIPLY FAC BY 10
D9F9	CC23	FINML6	X
DA04	CC2E	MUL10R	X
DA05	CC2F	TENC	FPB VALUE 10
DA0A	CC34	DIV10	DIVIDE FAC BY 10
DA13	CC3D	FDIVF	X
DA1B	CC45	FDIV	UNPACK MEMORY AND DIVIDE
DA1E	CC48	FDIVT	FAC = ARG/FAC
DA35	CC5F	DIVIDE	X
DA4B	CC75	SAV@UO	X
DA58	CC82	@SHFT	X
DA5B	CC85	SHFARG	X
DA69	CC93	DIVSUB	X
DA86	CCB0	LD100	X
DA8A	CCB4	DIVNRM	X
DA96	CCC0	DV0ERR	OVERFLOW ERROR VECTOR
DA9B	CCC5	MOVFR	MOVE RES TO FAC
DAAE	CCD8	MOVFM	MOVE MEMORY TO FAC
DAD3	CCFD	MOV2F	X
DAD6	CD00	MOV1F	X
DADC	CD06	MOVVF	X

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
DAE0	CD0A	MOVMF	MOVE FAC TO MEMORY
DB08	CD32	MOVFA	MOVE ARG TO FAC
DB0A	CD34	MOVFA1	X
DB0E	CD38	MOVFAL	X
DB18	CD42	MOVAF	MOVE FAC TO ARG
DB1B	CD45	MOVEF	X
DB1D	CD47	MOVAFL	X
DB26	CD50	MOVRTS	X
DB27	CD51	ROUND	ROUND FAC
DB2F	CD59	INCRND	X
DB37	CD61	SIGN	EXTRACT SIGN FROM FAC IN .A
DB3B	CD65	FCSIGN	X
DB3D	CD67	FCOMPS	X
DB44	CD6E	SIGNRT	X
DB45	CD6F	SGN	ROUTINE - SGN(X)
DB48	CD72	FLOAT	FLOAT THE SIGNED INTEGER IN FAC
DB50	CD7A	FLOATS	FLOAT THE SIGNED NUMBER IN FAC
DB55	CD7F	FLOATC	X
DB5B	CD85	FLOATB	X
DB64	CD8E	ABS	ROUTINE - ABS(X)
DB67	CD91	FCOMP	COMPARE ARG AND FAC .A=1←A<F
DB69	CD93	FCOMPN	X
DB9E	CDC8	FCOMPC	X
DBA4	CDCE	FCOMPD	X
DBA7	CDD1	@INT	FAC=INT(FAC) SIGNED ROUTINE - INT(X)
DBBB	CDE5	@ISHFT	X
DBC6	CDF0	@INTRT	X
DBC7	CDF1	@INT1	X
DBD8	CE02	INT	ROUTINE - INT(X)
DBF5	CE1F	CLRFAC	.A TO ALL POSISTIONS OF FAC
DBFE	CE28	INTRTS	X
DBFF	CE29	FIN	FBP INPUT, TXTPTR POINTS TO ASCII, RETURNS IN FAC
DC03	CE2D	FINZLP	X
DC12	CE3C	@PLUS	X
DC16	CE40	FINC	X
DC19	CE43	FINDG0	X
DC1B	CE45	FIN1	X
DC3A	CE64	FINEC1	X
DC3C	CE66	FINEC	X
DC3F	CE69	FNEDG1	X
DC41	CE6B	FINEC2	X
DC4D	CE77	FINDP	X
DC53	CE7D	FINE	X
DC55	CE7F	FINE1	X
DC5E	CE88	FINDIV	X
DC67	CE91	FINMUL	X
DC6E	CE98	FINONG	X
DC73	CE9D	NEGX@S	X
DC76	CEA0	FINDIG	X
DC7D	CEA7	FINDG1	X
DC8A	CEB4	FINLOG	X
DC9D	CEC7	FINEDG	X
DCAC	CED6	MLEX10	X
DCBA	CEE4	MLEXMI	X
DCBF	CEE9	N0999	FPB VALUE 99999999.90625
DCC4	CEEE	N9999	FPB VALUE 999999999.5

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
DCC9	CEF3	NMIL	FPB VALUE 10-9
0000	CEF8	CKSMC0	CHECKSUM BYTE $C000 ROM
DCCE	CF78	INPRT	PRINT CURRENT LINE NUMBER
DCD9	CF83	LINPRT	PRINT NUMBER IN (.A←HIGH .Y←LOW)
DCE6	CF90	STROU2	JMP STROUT
DCE9	CF93	FOUT	FPB OUTPUT
DCEB	CF95	FOUTC	X
DCF3	CF9D	FOUT1	X
DD0C	CFB6	FOUT37	X
DD15	CFBF	FOUT7	X
DD17	CFC1	FOUT4	X
DD22	CFCC	FOUT3	X
DD2D	CFD7	FOUT38	X
DD34	CFDE	FOUT9	X
DD3B	CFE5	FOUT5	X
DD3E	CFE8	BIGGES	X
DD53	CFFD	FOUTPI	X
DD54	CFFE	FOUT6	X
DD5F	D009	FOUT39	X
DD70	D01A	FOUT16	X
DD72	D01C	FOUT8	X
DD74	D01E	FOUTIM	CLOCK ENTRY INTO FOUT
DD76	D020	FOUT2	X
DD9A	D044	FOUT41	X
DD9C	D046	FOUT40	X
DDA3	D04D	FOUTYP	X
DDBE	D068	STXBUF	X
DDD0	D07A	FOULDY	X
DDD2	D07C	FOUT11	X
DDDF	D089	FOUT12	X
DDEF	D099	FOUT14	X
DDFB	D0A5	FOUT15	X
DE10	D0BA	FOUT19	X
DE13	D0BD	FOUT17	X
DE18	D0C2	FOUT20	X
DE1D	D0C7	FHALF	FPB VALUE 1/2
DE1F	D0C9	ZERO	X
DE22	D0CC	FOUTBL	TABLES OF POWERS OF -10↑X
DE46	D0F0	FDCEND	END OF POWERS TABLE
DE5E	D108	TIMEND	FPB TIME CONVERSION TABLES
DE5E	D108	SGR	ROUTINE - SGR(X)
DE68	D112	FPWRT	ROUTINE (ARG↑FAC)
DE71	D11B	FPWRT1	X
DE8B	D135	FPWR1	X
DEA1	D14B	NEGOP	NEGATE THE NUMBER IN FAC
DEAB	D155	NEGRTS	X
DEAC	D156	LOGEB2	FPB VALUE LOG(E) BASE 2
DEB1	D15B	EXPCON	LOG AND EXPONENT FPB TABLES
DED4	D184	EXP	ROUTINE - EXP(FAC)
DEEA	D194	STOLD	X
DEF5	D19F	GOMLDV	X
DEF8	D1A2	EXP1	X
DF08	D1B2	SWAPLP	X
DF2D	D1D7	POLYX	POLYNOMIAL EVALUATOR
DF43	D1ED	POLY	POLYNOMIAL EVALUATOR
DF47	D1F1	POLY1	X

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
DF56	D200	POLY3	X
DF5A	D204	POLY2	X
DF67	D211	POLY4	X
DF77	D221	RMULC	X
DF7B	D225	RADDC	X
DF7F	D229	RND	ROUTINE - RND(X)
DF9D	D247	QSETNR	X
DFB2	D25C	RND1	X
DFC2	D26C	STRNEX	X
DFD5	D27F	GMOVMF	X
DFD8	D282	COS	ROTINE - COS(X)
DFDF	D289	SIN	ROUTINE - SIN(FAC)
E011	D2BB	SIN1	X
E014	D2BE	SIN2	X
E021	D2CB	SIN3	X
E028	D2D2	TAN	ROUTINE - TAN(FAC)
E050	D2FA	COSC	X
E054	D2FE	PI2	FPB VALUE PI/2
E059	D303	TWOPI	FBP VALUE 2*PI
E05E	D308	FR4	FPB VALUE 1/4
E063	D30D	SINCON	SIN TABLES FPB VALUES
E08C	D32C	ATN	ROUTINE - ATN(FAC)
E094	D334	ATN1	X
E0A2	D342	ATN2	X
E0B5	D355	ATN3	X
E0BB	D35B	ATN4	X
E0BC	D35C	ATNCON	X
E0F9	D399	INITAT	BASIC SYSTEM INITIALIZATION CODE
E0FF	D39F	CHDGOT	X
E110	D3B0	CHDRTS	X
0000	D3B6	INIT	BASIC SYSTEM INITIALIZATION ROUTINE
E131	D3C9	MOVCHG	X
E15D	D400	LOOPMM	X
E165	D408	LOOPM1	X
E174	D417	USEDEC	X
E178	D41B	USEDEF	X
E1B7	D44B	WORDS	MESSAGE - 'BYTES FREE'
E1C4	D458	FREMES	MESSAGE - '### COMMODORE BASIC ###'
E1DE	D472	LASTWR	LAST BYTE OF BASIC SYSTEM CODE+1
0000	DEA4	PATCH2	PATCHES
E844	E844	CHTIM	X
0000	FF93	CONCAT	VECTOR - CONCAT
0000	FF96	DOPEN	VECTOR - DOPEN
0000	FF99	DCLOSE	VECTOR - DCLOSE
0000	FF9C	RECORD	VECTOR - RECORD
0000	FF9F	FORMAT	VECTOR - FORMAT
0000	FFA2	COLECT	VECTOR - COLLECT
0000	FFA5	BACKUP	VECTOR - BACKUP
0000	FFA8	DCOPY	VECTOR - COPY
0000	FFAB	APPEND	VECTOR - APPEND
0000	FFAE	DSAVE	VECTOR - DSAVE
0000	FFB1	DLOAD	VECTOR - DLOAD
0000	FFB4	DIRCAT	VECTOR - DIRECTORY
0000	FFB4	DCAT	VECTOR - CATALOG
0000	FFB7	RENAME	VECTOR - RENAME
0000	FFBA	SCRATC	VECTOR - SCRATCH

Tabelle D-3. Speicherauszug für CBM BASIC 4.0 (Forts.)

BASIC 3.0	BASIC 4.0	Labels	Description
0000	FFBD	READDS	VECTOR - DS AND DS$
FFC0	FFC0	COPEN	VECTOR - OPEN
FFC3	FFC3	CCLOS	VECTOR - CLOSE
FFC6	FFC6	CCOIN	VECTOR - SET INPUT DEVICE
FFC9	FFC9	CCOOUT	VECTOR - SET OUTPUT DEVICE
FFCC	FFCC	CLSCHN	VECTOR - RESTORE NORMAL I/O DEVICES
FFCC	FFCC	CCCHN	SAME AS ABOVE
C481	FFCF	INCHR	VECTOR - INPUT A CHARACTER (FROM SCREEN)
FFCF	FFCF	CINCH	SAME AS ABOVE
FFD2	FFD2	COUTCH	VECTOR - OUTPUT A CHARACTER
FFD5	FFD5	CLOAD	VECTOR - LOAD
FFD8	FFD8	CSAVE	VECTOR - SAVE
FFDB	FFDB	CVERF	VECTOR - VERIFY
FFDE	FFDE	CSYS	VECTOR - SYS
FFE1	FFE1	ISCNTC	VECTOR - TEST STOP KEY
FFE4	FFE4	CGETL	VECTOR - GET CHARACTER FROM KEYBOARD BUFFER
FFE7	FFE7	CCALL	VECTOR - ABORT ALL I/O CHANNELS
000F	0000	CONTW	Z
000D	0000	CNTWFL	Z
000F	0000	LINWID	Z
0010	0000	NCMWID	Z
006C	0000	STRNGI	Z
007F	0007	0	Z
C494	000U	INCRTS	Z
C721	0000	SNERRX	Z
D404	0000	FNDVAR	Z
D41E	0000	TVAR	Z
D427	0000	SVARS	Z
D433	0000	SVAR	Z
D43B	0000	SVARGO	Z
D440	0000	ARYVAR	Z
D48A	0000	ARYSTR	Z
D497	0000	DVARS	Z
D4A1	0000	DVAR	Z
D4B6	0000	DVAR2	Z
D4C0	0000	DVAR3	Z
D4DB	0000	GRBRTS	Z
D4E0	0000	GRBPAS	Z
D5B0	0000	FRETRT	Z
D745	0000	STORD1	Z
D745	0000	STORD1	Z
D745	0000	STORD1	Z
D745	0000	STORD1	Z

Anhang E

Literatur

Technische Daten (Kap. 1)

Druckschriften für alle CBM-Geräte sind bei den lokalen Commodore-Händlern erhältlich.
Niederlassungen der Commodore Business Machines, Inc. in Europa:

Deutschland	Commodore Büromaschinen GmbH Postfach 426 6078 Neu-Isenburg
Österreich	Ing. Ernst Steiner Hummelgasse 14 A-1130 Wien
Schweiz	Commodore AG Schweiz Dufourstr. 9 CH-4010 Basel

Meß- und Prüfsysteme (Kap. 1 und 7)

Grundlagen des IEC-Bus. BAF-Mitteilung. Vertrieb: EDOTRONIK GmbH, Sankt Veit Str. 70, 8000 München 80

Automatische Testsysteme mit CBM-Computern. Vertrieb: EDOTRONIK GmbH, Sankt Veit Str. 70, 8000 München 80

Checkliste für die Auswahl von IEC-Bus-Geräten. ELEKTRONIK-JOURNAL, INTERKAMA '80. Vertrieb: EDOTRONIK GmbH, Sankt Veit Str. 70, 8000 München 80

IEC-Bus/Interface/Tischrechner. ELEKTRONIK-PRAXIS, 6 (1975), Vertrieb: EDOTRONIK GmbH, Sankt Veit Str. 70, 8000 München 80

Schnittstellenkonverter CBM-Computer/IEC-625-Bus. Vertrieb: EDOTRONIK GmbH, Sankt Veit Str. 70, 8000 München 80

Textverarbeitung am Bildschirm (Kap. 3)

CBM-Wordcraft 80, Textverarbeitung mit dem Tisch-Computer. Druckschrift der CBM-Deutschland

AFS-TEXT 2.8 (bzw. 2.3). Textbearbeitung mit Formbriefausdruck für CBM 8032 (bzw. CBM 3001-Serie). Vertrieb: SM Softwareverbund Microcomputer GmbH, Scherbaumstr. 29, 8000 München 83

Programmieren in BASIC (Kap. 4)

Programming in BASIC for personal computers. D.L. Heiserman, Englewood Cliffs, Prentice Hall. 1981

Structured BASIC and beyond. W. Amsbury, Potomac, Md, Computer Science Press. 1980

Programme für CBM-Computer (Kap. 5)

Fertige Programme als Lösung von Aufgaben in Wirtschaft und Privatbereich werden u.a. in Form von Disketten angeboten. Nach Einlesen vom Floppy Disk-Speicher in den Computer eröffnen diese Programme einen Dialog mit dem Benutzer, in dessen Verlauf sie alle zur Aufgabe gehörenden Daten erfragen und die Lösung als Bildschirmdarstellung oder Ausdruck ausgeben.

CBM-Programme. Vom gleichen Hersteller der CBM-Computer-Systeme werden CBM-Programme für den deutschsprachigen Raum angeboten. Diese Programme liefern Lösungen in Aufgabenbereichen wie:

Textverarbeitung	Bauabrechnung
Datenverwaltung	Haustechnik
Planung u. Kalkulation	Baustatik
Finanzbuchhaltung	Immobiliengewerbe
Lohn- u. Gehaltsabrechnung	Statistik etc.

Druckschriften und Auskünfte sind über die Commodore-Händler erhältlich. (Siehe auch Adressen unter "Technische Daten").

SM-Programme. In Spezialisierung auf CBM-Computer werden durch den Softwareverbund Mikrocomputer, SM, Scherbaumstr. 29, 8000 München 83, Programme für folgende typische Anwendungsbereiche entwickelt und angeboten:

Textverarbeitung	Adressverwaltung
Fakturierung	Lagerverwaltung
Datenbanken	

Neben Einzelprogrammen auf Disketten (und teilweise Kassetten) gibt es Programmsammlungen für alle praktischen Anwendungsbereiche in Buchform:

Practical Basic Programs (englisch). Lon Poole, OSBORNE/McGraw-Hill. 1980. Vertrieb: te-wi-Verlag, Theo-Prosel-Weg 1, 8000 München 40

77 BASIC-Programme (deutsch). Lon Poole, te-wi-Verlag, München

Programmieren in ASSEMBLER (Kap. 7)

6502-Programmieren in Assembler. Lance L. Leventhal, te-wi-Verlag, München. 1981

Microcomputer-Technik (Kap. 7)

Einführung in die Microcomputer-Technik. Adam Osborne, te-wi-Verlag, München

Microcomputer-Grundwissen. Adam Osborne, te-wi-Verlag, München

IEEE-488-Bus/IEC-625-Bus (Kap. 7 und 1)

s.a. Literaturangaben unter "Meß- und Prüfsysteme"

IEC-Handbuch. Ein Handbuch für jeden IEEE-488-Gerätebesitzer mit Standard-Bus. M.P. Gottlob, Hofacker, München. 1980

PET and the IEEE 488 Bus (GPIB). E. Fisher and C.W. Jensen, OSBORNE/McGraw-Hill. 1980. Vertrieb: te-wi-Verlag, München

Grundlagen des IEC-Bus. BAF-Mitteilung 1981. Vertrieb: Softwareverbund Mikrocomputer, Scherbaumstr. 29, 8000 München 83

Schnittstellenkonverter CBM-Computer/EC-625-Bus.

EDOTRONIK GmbH, Sankt Veit Str. 70, 8000 München 80

IEC-Bus/Interface/Tischrechner. ELEKTRONIK-PRAXIS, 6 (1975). Vertrieb: EDOTRONIK GmbH, Sankt Veit Str. 70, 8000 München 80

IEC-625-BUS. Deutsche Norm DKE 66.22(2)

Speicherauszüge mit deutschem Kommentar (Anhang D)

SM-ROM-Listing 8000. Kommentiertes ROM-Listing des CBM 8032. Vertrieb: SM Softwareverbund Microcomputer, Scherbaumstr. 29, 8000 München 83

SM-ROM-Listing 3000. Kommentiertes ROM-Listing der CBM 3001-Serie. Vertrieb: SM Softwareverbund Microcomputer, Scherbaumstr. 29, 8000 München 83.

Weil jeder 2. verkaufte Tischcomputer in Deutschland uns zur Nr. 1 gemacht hat, kommen auch Sie zu Commodore.

Praktisch ist heute jeder 2. kommerziell genutzte Mikrocomputer in Deutschland ein Commodore. Dieser Erfolg kommt nicht von ungefähr. Die zuverlässige Technik, der geringe Installationsaufwand, das komplette Systemangebot und das günstige Preis/Leistungs-Verhältnis sind der Grund dafür.

Dazu beigetragen haben aber auch die vielen auf die Praxis zugeschnittenen Software-Lösungen. Von Finanz-Buchhaltung über Lohn-und Gehaltsabrechnung bis hin zur Planung, Kalkulation, Statistik und Textverarbeitung. Jetzt hat Commodore die Erfolgsserie CBM 8000 ergonomisch optimiert und modernen arbeitsphysiologischen Gesichtspunkten angepaßt. Schwenkbarer Bildschirm und abnehmbare Tastatur kennzeichnen den neuen Arbeitsplatzcomputer. Ein weiterer Grund, sich für Commodore zu entscheiden.

***Vertragshändler-Anschriften bei Commodore-Vertriebsbüros: Hamburg** · Eiffestr. 600 · Tel. (040) 211386. **Düsseldorf** · Fleher Str. 28 · Tel. (0211) 312047/48. **Frankfurt** · Lyoner Str. 38 · Tel. (0611) 6638-199. **Stuttgart** · Olgastr. 77 · Tel. (0711) 247329. **München** · 8013 Haar · Hans-Pinsel-Str. 9–10 · Tel. (089) 463009. **Schweiz:** Commodore AG · Aeschenvorstadt 57 · **CH-4010 Basel** · Tel. (004161) 237800. **Österreich:** Commodore · Fleschgasse 2 · **A-1130 Wien** · Tel. (0043222) 8274720.*

Ihr freundlicher Computer-Kontakt-Coupon

Ich interessiere mich für CBM Tischcomputer

Name: _____

Anschrift: _____

Einsenden an: Commodore GmbH
Abt. CBM · Lyoner Str. 38
6000 Frankfurt 71

commodore COMPUTER

MIKROCOMPUTER–GRUNDWISSEN

Eine allgemeinverständliche Einführung in die Mikrocomputer-Technik – vom Mikrocomputer-Papst Dr. Adam Osborne.
Optimal als Einstieg für Elektronik-Laien; zur Kontaktaufnahme mit diesem die Technik und unsere Umwelt revolutionierenden Gebiet.
Dieses Buch ist aber kein trockener Lernstoff, sondern in seiner didaktisch ausgezeichneten und humorvollen Form – mit vielen Illustrationen und Cartoons – ein Werk, das man verschlingt. Und dabei spielend leicht ein Thema begreift, das unser Leben immer mehr beherrscht.

MIKROCOMPUTER–GRUNDWISSEN – von Adam Osborne
304 Seiten, Paperback, DM 36,– inkl. 6,5% MwSt., zuzüglich Versandspesen

---

te-wi Verlag GmbH
Telefon 089/19 20 90
⟨129 2090⟩

te-wi

Theo-Prosel-Weg 1
8000 München 40

weiterführende Literatur und Software

M&T-Neuerscheinungen 1983!

Ch. Langfelder

BASIC ohne Probleme Band 2: Übungen

20 ausgewählte Routinen und Programme zum Veranschaulichen und Üben allgemeiner Programmiertechniken auf CBM-Rechnern (CBM 8032)

Die Programme setzen sich aus einer Beschreibung, Bedienungsanweisungen, Beispielen, Anmerkungen und reichlich mit Kommentaren versehenen Programmprotokollen zusammen. Die Information wird, sofern sinnvoll und erforderlich, durch Tips, Erläuterungen und Flußdiagramme ergänzt. Die Programme haben in erster Linie die Aufgabe, allgemeine Programmiertechniken zu veranschaulichen. Die Betonung liegt auf den alltäglichen und daher besonders wichtigen Problemen in der Programmierung: die Behandlung und Verarbeitung von Eingabedaten, die Steuerung des Cursors, die Verwendung von Unterprogrammen und die Programmierung von Fehlerausgängen.

1983, 119 Seiten

Bestellnummer MT 490 **DM 24,—***

BASIC ohne Probleme Band 1: Unterweisung

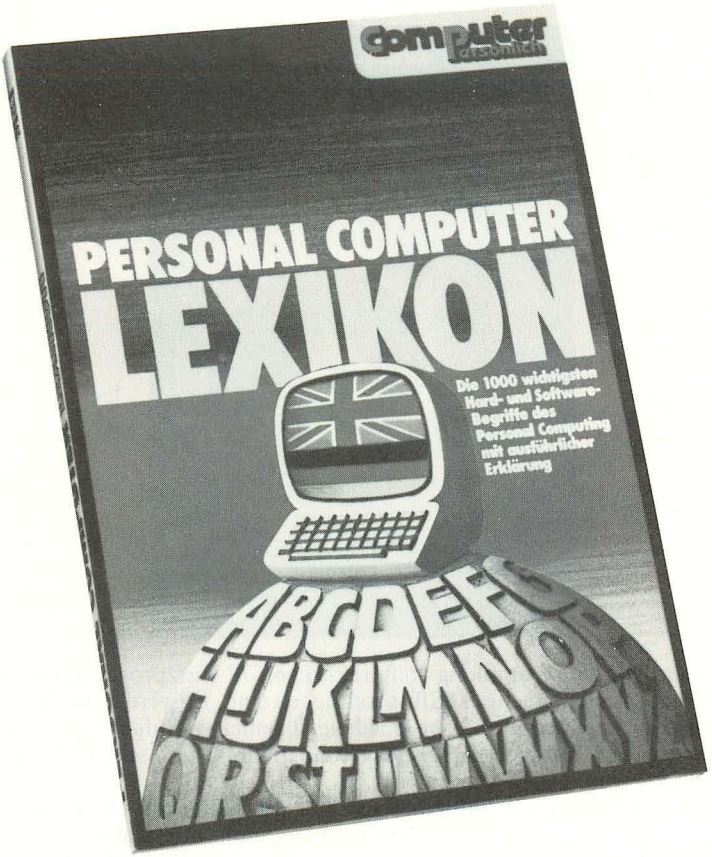
Vom selben Autor erscheint im Februar 1983 eine Einführung in BASIC mit CBM-Rechnern (ca. 170 Seiten).
In 12 Kapiteln wird der Leser Schritt für Schritt mit BASIC, dem CBM-Rechner und seiner Bedienung vertraut gemacht. Jedes Kapitel schließt mit Übungen und Aufgaben ab — als Kontrolle für den jeweiligen Wissensstand. Im Anhang befinden sich dann unter anderem die Lösungen, ein Glossar, ein Stichwortregister usw.

Bestellnummer MT 480 **DM 29,80***

* Die Preise verstehen sich inkl. MwSt. zuzügl. Versandkosten.

Markt&Technik

Hans-Pinsel-Str. 2, 8013 Haar bei München, Telefon 089/4613-225



Dieses Lexikon wurde entwickelt, um die Welt der Personal Computer transparenter zu machen. Es enthält die über 1000 wichtigsten Hard- und Software-Begriffe des »Personal Computing« und verwandter Gebiete. Alle Begriffe werden auf deutsch erklärt. Zusätzlich wird die englische Übersetzung des deutschen Suchbegriffes angegeben. Wichtig: Im Anhang befindet sich ein Register englisch-deutsch. So gibt es auch beim Lesen englischsprachiger Dokumentationen keine Probleme.

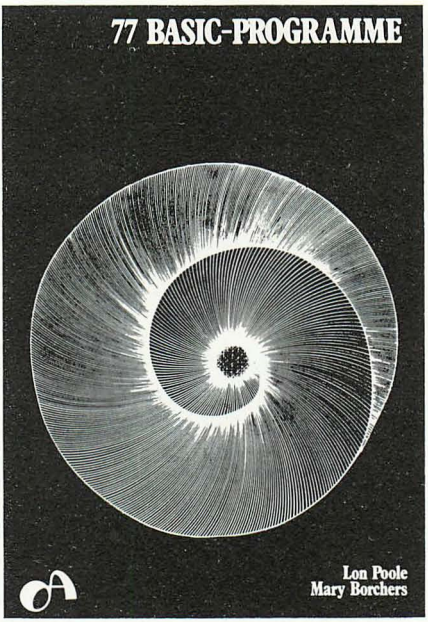
1982, 136 Seiten, Register: englisch-deutsch
Bestellnummer MT 390 DM 19,80*

* inkl. MwSt. zuzügl. Versandkosten

Markt&Technik

Geschäftsbereich Fachliteratur
Hans-Pinsel-Str. 2, 8013 Haar bei München
Telefon 089/4613-225

77 BASIC–PROGRAMME



Dieses Buch enthält eine Sammlung von praktischen 77 Kurzprogrammen, die in der populären Programmiersprache BASIC geschrieben sind. Es werden mathematische, finanztechnische, statistische und verschiedene allgemeine Aufgaben behandelt.
Die leichtverständlich erläuterten Beispiele können direkt und selbst ohne jede Erfahrung in BASIC verwendet werden.
Jedes Programm wird zunächst kurz beschrieben und später mit Hinweisen auf Änderungsmöglichkeiten der entsprechenden Programme aufgeführt.
Die Programme haben einen Umfang von etwa 15 bis 130 Zeilen BASIC.
Das Buch entstand unter Verwendung eines CBM-Computers und ist somit eine besonders nützliche Hilfe für jeden Besitzer dieses beliebten Computers.
Jedoch lassen sich die Programme auch ohne große Änderungen und bei einiger Sorgfalt auf anderen bekannten Systemen verwenden.
Der Ausdruck erfolgte auf einem Heathkit-Drucker WH-14. Bei der Verwendung anderer Drucker mit unterschiedlicher Zeilenlänge können die Print-Anweisungen innerhalb der Programme entsprechend geändert werden.
Die Darstellung der Programme ist ebenso auf einem Bildschirm möglich.

77 BASIC-Programme – Ein Buch das Spaß am Nutzen bringt.
von Lon Poole und Mary Borchers
208 Seiten, Softcover, DM 39,– inkl.6,5% MwSt., zuzüglich Versandspesen

---

te-wi Verlag GmbH
Telefon 089/19 20 90
⟨129 2090⟩

te-wi

Theo-Prosel-Weg 1
8000 München 40

PET/CBM VC 20 Besitzer

Kennen Sie **SYNTAX** – das Programm-Magazin auf Kassette?

Es bringt jeden Monat 5 neue Programme in deutscher Sprache aus allen Bereichen. Zum Beispiel Datei-Systeme, Textverarbeitung, Lehrgang Maschinen-Sprache, User-Programm usw.

Kenner der **SYNTAX-MAGAZINE** loben Leistung und Preis.

Seit Januar 1982 gibt es das **SYNTAX**-Programm-Magazin auf Kassette auch für den VC 20.

Fordern Sie gleich heute noch kostenlose Informationen unter Angabe Ihres Gerätetyps von ....

SYNTAX

Soft- u. Hardware GmbH

P. B. 1609, 7550 Rastatt
Tel. 07222/3 42 96

NEU **CBM 8000**

COMGRAPH-8000 Highspeed-Grafik mit Soft-Pen, 130.000 Bildp. **999,–**
ROM-KARTEN erweit. auf 2 x 7 Steckplätze **282,–**
auf 1 x 3 Steckplätze **151,–**
auf 1 x 2 Steckplätze **128,–**
ITOH-DRUCKER Preise auf Anfrage.
EXBASIC/L.2 Super-Utilitys in 2 Eprom **380,–**
80/40-EPROM 3000-Ba. + 4000-Ba. u. Ma. Prg. laufen auf jedem CBM-8000 **211,–**
PETSPEED Optimierender BASIC-Compiler (!!!) **1243,–**
CBM-LEITFADEN/LEHRGANG. 1000 Tips + Tricks ca. 320 DinA4-Seiten **299,–**
DATEX-2000 für sehr anspruchsvolle Textverarbeitung! **954,–**
LIST-SCHUTZ + AUTOM. START Ihr Programm zur Wahrung Ihres Copyrights **359,–**
CBM-64 + gesamte Peripherie auf Anfrage!

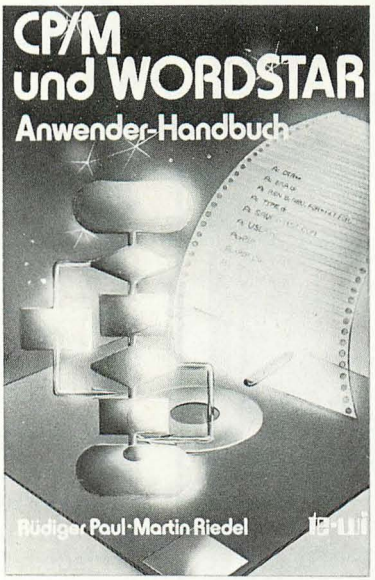
Viel Software + Hardware auf Anfrage in unserer Info!

Porto + Verp., je Lieferung 8,–/10,–.
Im Versand nur Nachn. o. Vorkasse!
Preise **INKL.** 13% MwSt.

Fordern Sie unsere INFO an!

WCS-WEMPER COMPSOFT!
LOHER STRASSE 57
2000 HAMBURG 73
TEL.: (040) 6 72 65 36

CP/M und WORDSTAR



Orientiert man sich im Computerbereich an Systemen amerikanischer Herkunft, die die Anwender-Zielgruppe unterer und teilweise mittlerer Bereich betrifft, stößt man immer wieder auf CP/M.
Für dieses Software-System zum Betrieb von Computern ist nun ein Standardwerk in deutscher Sprache erschienen, das dem ständig steigenden Kreis von Mikrocomputer-Benutzern eine fundamentale Einarbeitungshilfe bietet. Dieses Buch baut auf den ursprünglichen Unterlagen über CP/M von Digital Research einerseits auf, setzt diese jedoch nicht voraus. Es werden die grundsätzlichen Zusammenhänge des BIOS beschrieben, wie auch das Mehrbenutzer-Betriebssystem MP/M, das auf CP/M aufbaut. Als logische Vervollständigung rundet die Beschreibung über das Textsystem „WordStar" den gebotenen Inhalt sinnvoll ab.
Für alle, die das Riesen-Reservoire an durchweg kommerzieller Software beanspruchen und darüberhinaus selbst Programme entwickeln wollen.

CP/M und WORDSTAR – von Rüdiger Paul und Martin Riedel
144 Seiten, Paperback, DM 29,80 inkl. 6,5% MwSt., zuzüglich Versandspesen

---

te-wi Verlag GmbH
Telefon 089/19 20 90
⟨1292090⟩

te-wi

Theo-Prosel-Weg 1
8000 München 40

. . . und jetzt wird Ihr CBM grafikfähig:

COMGRAPH 8000 professional-

High-Resolution Grafik für den CBM 8032/8096

- 128.000 einzeln setz- und löschbare Bildpunkte
- Text und Grafik beliebig mischbar
- GBASIC 8000 – die leistungsfähige und komfortable Software mit über 30 echten Grafik-Basic-Befehlen und vielen Systemroutinen
- Kompakter Aufbau, lötfreier Anschluß!
- Zuverlässig in vielen hundert Installationen
- Eingebauter SOFT-PEN, zum schnellen Zeichnen und Korrigieren direkt auf dem Bildschirm!

Ohne Vorkenntnisse erstellen Sie dank des einmalig komfortablen Grafik-BASICs (GBASIC 8000) spielend komplexe Grafiklösungen wie Funktionsdarstellungen, Diagramme, Statistiken und Formulare, um nur einige Anwendungen zu nennen.

Die Grafikbefehle sind zusätzlich zum Standard-CBM-BASIC verfügbar und können direkt benutzt werden.

Grafik-Befehle:

plot	plot #	move	move #	erase
erase #	grid	clear	blank	display
ellipsis	ellipsis #	point	where	lplot
initialize	xmap	ymap	axis	hardcopy
pen	setpen	chrplot	chrdir	gload
gsave	hex $	dec	kill	help

Das komplette Grafikpaket inklusive Demo-Diskette erhalten Sie zum Preis von nur 898,–DM inkl. MwSt. bei Ihrem CBM-Fachhändler oder direkt vom Hersteller.

Übrigens:
Wir liefern eine breite Palette an CBM-Zubehör – informieren Sie sich durch unsere ausführlichen kostenlosen Unterlagen!

RÖSCH DATENTECHNIK

Software- und Systementwicklung
Industrieelektronik

Steintorwall 12 Tel.: 0531 / 44 200
D-3300 Braunschweig Tlx.: 952759 rodat-d

CHIP. Das Mikrocomputer-Fachmagazin. Für alle, die mehr wissen wollen und mehr wissen müssen, wenn es um Mikrocomputer geht. In Hobby und Beruf. Denn **CHIP** bringt umfassend und verständlich alles worauf es ankommt: Hintergrundberichte, Beispiele, umfassende Anleitungen, Software, Nachrichten, Trends und Meinungen. Ausgabe für Ausgabe eine geballte Ladung an Wissenswertem und Wichtigem. Damit Sie Bescheid wissen und mitreden können.

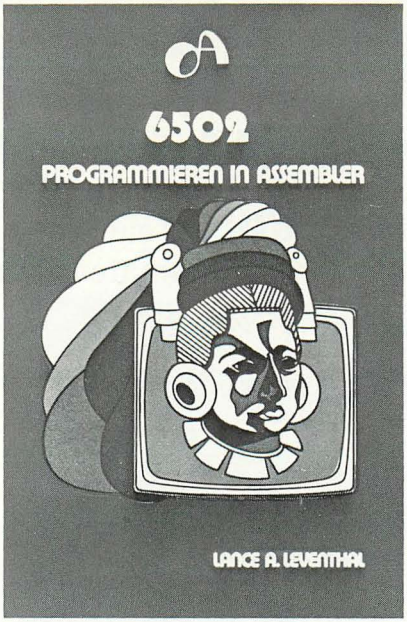
CHIP gibt es monatlich im Zeitschriften- und Buchhandel. Oder einfach bestellen beim **CHIP**-Leserservice, Vogel-Verlag Postfach 67 40, D-8700 Würzburg 1

CHIP SPECIAL

Die umfangreichen Sonderpublikationen von CHIP zu den aktuellsten und wichtigsten Themen aus der Welt der Mikrocomputer. Jedes **SPECIAL** für ganze 24 Mark! Fordern Sie ausführliches Informationsmaterial an.

Die **CHIP SPECIALS** gibt es beim Zeitschriften-, Buch- und Elektronikfachhandel. Oder direkt bestellen bei CHIP, Postfach 67 40, D-8700 Würzburg 1.

6502 – PROGRAMMIEREN IN ASSEMBLER



Wenn Sie den Mikroprozessor 6502 – das "Herzstück" vieler Computersysteme und vor allem des CBM-Computers selbst in der Assemblersprache programmieren wollen, ist dies das richtige Buch für Sie.
Dieser Band, der eine ganze Serie von deutschsprachigen Büchern über die Programmierung in Assembler repräsentiert, enthält über 80 praktische Programmierbeispiele im Standardformat einschließlich Flußdiagrammen, Quellprogrammen, Objektcodes und erläuternden Texten.
Jeder Befehl des Mikroprozessors 6502 wird detailliert erklärt.

Weitere Themen u.a.:
- Assembler-Vereinbarungen des 6502
- Programmierung des Unterbrechungsprogramms des 6502
- E/A-Bausteine und Interfacing-Verfahren des 6502

Ausführliche Besprechungen für die Erstellung von Programmen, von der Definition der Aufgabe, über Testen, Fehlersuche, Dokumentation, bis hin zu modularer und strukturierter Programmierung runden dieses aktuelle Werk ab.
Die Darstellung ist außerordentlich gründlich, und didaktisch gut aufbereitet.

6502 – PROGRAMMIEREN IN ASSEMBLER – Ein unentbehrliches Hilfsmittel für den Hardware-Praktiker, das auch dem Einsteiger ein leichtes Einarbeiten bietet.
von Lance A.Leventhal
700 Seiten, Paperback, DM 59,– inkl.6,5% MwSt., zuzüglich Versandspesen

---

te-wi Verlag GmbH
Telefon 089/192090

te-wi

Theo-Prosel-Weg 1
8000 München 40

Helmut Rennen
GmbH & Co. KG.
Büro-Organisation + Datenverarbeitung

Wir haben ein großes Sortiment an Hardware und Dienstleistungen anzubieten. Überzeugen Sie sich und besuchen Sie uns, selbst der weiteste Weg lohnt sich.

BERATUNG	Wir beraten Sie gerne in der Auswahl des richtigen Systems, vom Microcomputer bis zum Mehrplatzsystem, ob kaufmännische Anwendung oder technisch-wissenschaftliche Aufgaben.
SCHULUNG	Anwender-Seminare für Standardprogramme, Programmierschulung in Basic, Assembler, usw.
PROGRAMME	Standardprogramme von A–Z.
PROGRAM-MIERUNG	Wir programmieren Ihre speziellen Wünsche und helfen bei der Systemanalyse.
TECHNISCHER KUNDENDIENST	Wir bieten sachkundigen, technischen Service und interessante Wartungsverträge.

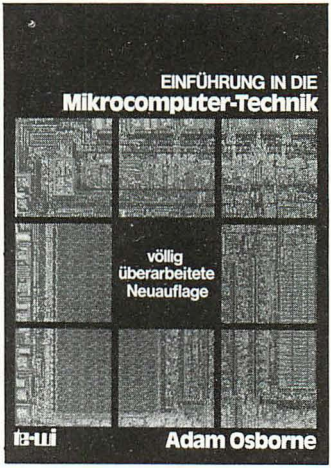
Ein kleiner Auszug aus unserem weiteren Verkaufsprogramm:
Adcomp – Commodore – Epson – Hewlett Packard – Itoh
IBM – 3M – Sirius – Watanabe.

Einer der größten Commodore Fachhändler in Deutschland mit Sitz in
4000 Düsseldorf, Martinstraße 55, Telefon (0211) 306090
4300 Essen, Kleiststraße 2, Telefon (0201) 237139

Helmut Rennen
GmbH & Co. KG.
Büro-Organisation + Datenverarbeitung

EINFÜHRUNG IN DIE MIKROCOMPUTER-TECHNIK

VÖLLIG NEU ÜBERARBEITET!



Wie konnte dieses Buch Bestseller in den USA werden? Nicht nur die international anerkannte Kompetenz des Autors, Adam Osborne, auf dem Gebiet der Mikrocomputer-Technik gibt die Erklärung – sondern seine ungewöhnliche Fähigkeit, eine der bedeutsamsten Technologien unserer Zeit mehr in Bildern statt Worten erklären zu können. Beim Durchblättern dieses Buches werden Sie kaum einen Gedanken finden, der nicht als Bild anschaulich gemacht wurde. Diese Einführung in Prinzipien, Funktion und Systemaufbau von Mikrocomputern vermittelt Ihnen eine lebendige und umfassende Vorstellung vom Wesen dieser Technik und befähigt Sie, konkrete Aufgaben durch richtige Fragen an Experten oder durch Verständnis der Systembeschreibungen von Mikrocomputern zu lösen. Sie werden nicht nur die Komponenten eines Mikrocomputersystems richtig beurteilen und für Ihre Zwecke geeignet auswählen können, sondern auch zu Fragen der Programmierung, Vergleich alternativer Mikrocomputersysteme, Fragen der Wirtschaftlichkeit bei Anwendung in Massenprodukten und zur Beurteilung zukünftiger Entwicklungen mit Sachverstand Stellung nehmen können. Dieses bestens bekannte Standardwerk des Autors, das auch an über 500 Universitäten weltweit als reguläre Studiengrundlage eingeführt ist, hat eine absolute Überarbeitung erfahren. Es spiegelt den allerletzten Stand der faszinierenden Mikrocomputer-Technik wider.

EINFÜHRUNG IN DIE MIKROCOMPUTER-TECHNIK – GRUNDLAGENBUCH
4. völlig überarbeitete Neuauflage
von Adam Osborne
488 Seiten, zahlreiche Abbildungen, Hardcover, farbiger Schutzumschlag,
DM 66,– inkl. 6,5% MwSt., zuzüglich Versandspesen

---

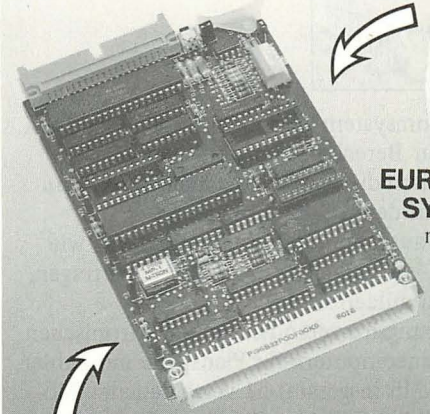
te-wi Verlag GmbH
Telefon 089/19 20 90
<1 29 20 90>

te-wi

Theo-Prosel-Weg 1
8000 München 40

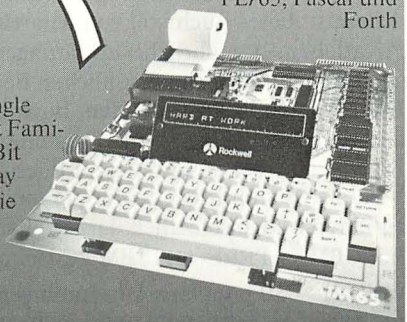
Rockwell International

ENTWICKLUNGSSYSTEM RDC
für alle Rockwell Mikroprozessoren
und Computer SBC – und RM 65- Module

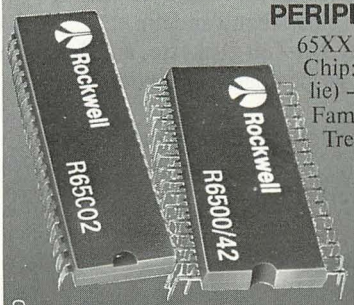


EUROPAKARTEN
SYSTEM RM 65
mit 6502 CPU für industrielle
Anwendungen

EINPLATINEN
COMPUTER AIM 65
mit Assembler, Basic,
PL/65, Pascal und
Forth



CPU UND
PERIPHERIE
65XX, 68000, Single
Chip: 6511 (8 Bit Fami-
lie) – PPS/4 (4 Bit
Familie), Display
Treiber- Familie

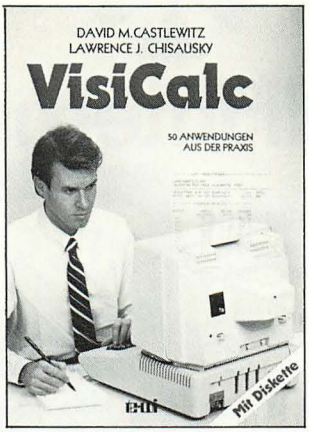


tm 3440

bitronic

BITRONIC GMBH · EINSTEINSTRASSE 127 · 8000 MÜNCHEN 80 · TELEFON 0 89/4 70 20 98 · TELEX 5 212 931 BIT D
ZWEIGBÜRO STUTTGART: DIESELSTR. 30 · 7016 GERLINGEN · TELEFON 0 71 56/2 40 51 · TELEX 7 266 743 BIT D
ZWEIGBÜRO DORTMUND: KARL-MARX-STR. 59 · 4600 DORTMUND 1 · TELEFON 02 31/52 82 93 · TELEX 8 22 664 BIT D

VisiCalc – 50 populäre Anwendungen
– Die heutige Art, Daten transparent zu machen –



Dieses Buch befaßt sich mit dem Programmsystem VisiCalc, mit dem Sie Ihren Heimcomputer in einen "Elektronischen Berechnungsbogen" verwandeln können. Damit ist gemeint, daß elektronisch die Ausführung aller Berechnungen und die Darstellung der Ergebnisse auf dem Bildschirm erfolgt.
Und "Berechnungsbogen" steht für alle ausgebreiteten Zahlenwerke – wie etwa ein Formular zur Steuererklärung – in das nicht nur Zahlen einzutragen, sondern auch Summen, Überträge usw. zu bilden sind.
VisiCalc läßt die flüchtigen, fast verzögerungslos veränderbaren elektronischen Berechnungen und Darstellungen auf Heimcomputern für Planspiele mit Zahlen oder "Was-Wenn-Überlegungen" nutzen. Mit insgesamt 51 verschiedenen Modellen ist dieses Buch voll von derartigen Anwendungen in Wirtschaft und Privatleben. Ausführliche Erklärungen und Auflistungen machen es leicht, alle Programme nach persönlichen Bedürfnissen selbst abzuwandeln.
Der besondere Clou: eine dem Buch beigefügte 5 1/4-Zoll-Diskette, auf der alle Berechnungen und die Darstellungsformen gespeichert sind, erleichtern Ihnen das Arbeiten mit diesem Buch.
VisiCalc ist zum Inbegriff für die moderne Art geworden, Berechnungen auszuführen und anschaulich darzustellen.
Für CBM-, APPLE- und IBM-PC-Heimcomputer erhältlich.

VisiCalc – 50 populäre Anwendungen
von David M. Castlewitz und Lawrence J. Chisausky
184 Seiten, Paperback, Diskette
DM 69,– inkl. 6,5% MwSt., zuzüglich Versandspesen

---

te-wi Verlag GmbH
Telefon 089/19 20 90
⟨129 20 90⟩

te-wi

Theo-Prosel-Weg 1
8000 München 40

Für alle CBM Besitzer oder die, die es werden wollen, bietet diese einzigartige Fundgrube eine schrittweise Einführung bis hin zur professionellen Ausnutzung aller Möglichkeiten des beliebten Computers. Ein Nachschlagewerk, das einem hilft, seinen Computer erst richtig zu verstehen.

commodore
Approved

Weitere deutsche Bücher aus dem te-wi Verlag

Einführung in die Mikrocomputer-Technik
(A. Osborne)
Das schon legendäre Standardwerk des Autors wurde in dieser Ausgabe restlos überarbeitet. Es spiegelt den allerletzten Stand dieser bedeutenden Technologie wieder. Geblieben ist aber die Osborne-typische Darstellung, in der auch vielleicht komplizierte Vorgänge bildhaft verständlich gemacht werden. Wer dieses Buch gelesen hat, kennt sich in der Welt der Mikrocomputer-Technik aus.

Treibjagd – Im Strudel der Mikroelektronik
(A. Osborne)
Ein populär geschriebenes Werk aus der Sicht des Insiders Osborne.

Mikrocomputer-Grundwissen (A. Osborne)
Eine allgemeinverständliche Einführung in die Mikrocomputer-Technik – optimal als Einstieg für Elektronik-Laien.

8080A/8085 – Programmieren in Assembler
6800 – Programmieren in Assembler
6502 – Programmieren in Assembler
(Lance A. Leventhal)
Drei Titel über die Assemblersprachen-Programmierung. Sie sind für den Hardware-Praktiker ein unentbehrliches Hilfsmittel und ermöglichen dem Einsteiger ein leichtes Einarbeiten in die Assemblersprache.

16 Bit Generation – Z8000
(Peter Stuhlmüller)
Eine ausführliche Information über den grundlegenden Aufbau sowie die Funktion (hard- und softwaremäßig) der CPU. Die sehr umfangreiche Hardware-Darstellung gipfelt in der Beschreibung eines kapitalen Parallelrechner-Systems.

Das 8086/8088 Buch
(Russell Rector/George Alexy)
Dieses Buch wendet sich an die Leser, die mit Personalcomputern mit einer 8086- oder 8088 CPU experimentieren, und auch an Techniker, die sich mit der Entwicklung komplexer Hardware befassen.
In Vorbereitung 1983: 68000, 16000, UNIX

CP/M und WordStar Anwender Handbuch
(Rüdiger Paul/Martin Riedel)
Für das CP/M-Software-System ist dieses Standardwerk in deutscher Sprache erschienen, das dem ständig steigenden Kreis von Mikrocomputer-Benutzern eine fundamentale Einarbeitungshilfe bietet.

VisiCalc
(David M. Castelwitz/Lawrence J. Chisausky)
VisiCalc ist zum Inbegriff für die moderne Art der Berechnung und anschaulich-übersichtliche Darstellung von Daten auf Heimcomputern geworden. Dieses Buch enthält eine Sammlung der 50 häufigsten VisiCalc-Anwendungen in Wirtschaft und Privathaushalten. Ausführliche Erklärungen und Auflistungen machen es leicht, alle Programme nach persönlichen Bedürfnissen selbst abzuwandeln.

77 BASIC-Programme
(Lon Poole/Mary Borchers)
Dieses Buch beschreibt 77 Kurzprogramme, die finanztechnische, mathematische, statistische und verschiedene allgemeine Aufgaben mit Programmbeispielen behandeln.

ISBN 3-921803-13-6